eCMD C/C++ Dll Reference Manual

Generated by Doxygen 1.3.5

Wed Jan 26 11:26:29 2005

Contents

Chapter 1

eCMD C/C++ Dll Main Page

1.1 Introduction

Common Hardware Access Programming Interface (eCMD)

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

1.2 eCMD Core Include Files

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

- **ecmdClientCapi.H**(p. 103)
- **ecmdDataBuffer.H**(p. 157)
- **ecmdStructs.H**(p. 175)
- **ecmdReturnCodes.H**(p. 161)
- **ecmdUtils.H**(p. 185)
- **ecmdSharedUtils.H**(p. 172)

1.3 Link objects

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

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

To create Linux x86 binaries, the following is required:

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

1.4 eCMD Extensions

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

1.4.1 CIP (Cronus/IP) Extension

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

Include files :

- **cipClientCapi.H**(p. 89)
- **cipStructs.H**(p. 98)

1.4.2 Cronus Extension

This extensions provides Cronus only interfaces.

Include files :

- **croClientCapi.H**(p. 99)
- **croStructs.H**(p. 102)

1.5 DLL Version

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

1.6 The ecmdDataBuffer class

Data is passed between the client and the DLL with the **ecmdDataBuffer**(p. 25) class. The **ecmdDataBuffer**(p. 25) object is linked on both the client side and the DLL side.

The **ecmdDataBuffer**(p. 25) 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. 25) 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. 25) conversion-to-string call.

1.7 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

1.7.1 Aix

```
# Choose the eCMD Release to build against
ECMD_RELEASE := $CTEPATH/tools/ecmd/rel


testclient: testclient.o ${ECMD_RELEASE}/capi/ecmdClientCapi_aix.a
xlC -+ -g -brtl -L${ECMD_RELEASE}/lib -lecmd_aix testclient.o ${ECMD_RELEASE}/capi/ecmdClientCapi_aix.a -o test


testclient.o: testclient.C ${ECMD_RELEASE}/capi/ecmdClientCapi.H ${ECMD_RELEASE}/capi/ecmdDataBuffer.H ${ECMD_R
xlC -+ -g -c -I${ECMD_RELEASE}/capi/ testclient.C -o testclient.o
```

1.7.2 Linux x86

```
# Choose the eCMD Release to build against
ECMD_RELEASE := $CTEPATH/tools/ecmd/rel


testclient.linux: testclient_linux.o dll/ecmdClientCapi_x86.a
g++ -g -ldl -L${ECMD_RELEASE}/lib -lecmd_x86 testclient_linux.o ${ECMD_RELEASE}/capi/ecmdClientCapi_x86.a -o te


testclient_linux.o: testclient.c ${ECMD_RELEASE}/capi/ecmdClientCapi.H ${ECMD_RELEASE}/capi/ecmdDataBuffer.H ${
g++ -g -c -I${ECMD_RELEASE}/capi/  -ftemplate-depth-30 testclient.c -o testclient_linux.o
```

1.8 Example

```
&#35;include
&#35;include


&#35;include $<$ ecmdClientCapi.H (p. 103)$>$
&#35;include $<$ ecmdDataBuffer.H (p. 157)$>$


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


  // A buffer to store our data
   ecmdDataBuffer (p. 25) data;
  uint32_t rc = 0;
  // This is the chip target to operate on
   ecmdChipTarget (p. 19) 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. 126);

// 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. 126);
  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. 69) 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. 113);

  return rc;

}
```

Chapter 2

eCMD C/C++ Dll Class Index

2.1 eCMD C/C++ Dll Class List

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

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:

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.

- std::string **clockDomain**

  Clock domain this array belongs to.

- **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 std::string ecmdArrayData::clockDomain

Clock domain this array belongs to.

4.1.2.6 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 Member Functions

- **ecmdCageData** ()
- **~ecmdCageData** ()
- uint32_t **flatten** (uint8_t *o_buf, uint32_t &i_len)
- uint32_t **unflatten** (const uint8_t *i_buf, uint32_t &i_len)
- uint32_t **flattenSize** (void)
- void **printStruct** (void)

Public Attributes

- uint32_t **cageId**

  (Detail: Low) Cage number of this entry

- uint32_t **unitId**

  (Detail: High) Unit Id of this entry

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

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

4.3.1 Detailed Description

Used for the ecmdQueryConfig function to return cage data.

Operators Supported : <

4.3.2 Constructor & Destructor Documentation

4.3.2.1 ecmdCageData::ecmdCageData () [inline]

4.3.2.2 ecmdCageData::~ecmdCageData () [inline]

4.3.3 Member Function Documentation

4.3.3.1 uint32_t ecmdCageData::flatten (uint8_t * *o_buf*, uint32_t & *i_len*)

4.3.3.2 uint32_t ecmdCageData::unflatten (const uint8_t * *i_buf*, uint32_t & *i_len*)

4.3.3.3 uint32_t ecmdCageData::flattenSize (void)

4.3.3.4 void ecmdCageData::printStruct (void)

4.3.4 Member Data Documentation

4.3.4.1 uint32_t ecmdCageData::cageId

(Detail: Low) Cage number of this entry

4.3.4.2 uint32_t ecmdCageData::unitId

(Detail: High) Unit Id of this entry

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

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

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

- **ecmdStructs.H**

4.4 ecmdChipData Struct Reference

Used for the ecmdQueryConfig function to return chip data.

`#include <ecmdStructs.H>`

Public Member Functions

- **ecmdChipData** ()
- **~ecmdChipData** ()
- uint32_t **flatten** (uint8_t *o_buf, uint32_t &i_len)
- uint32_t **unflatten** (const uint8_t *i_buf, uint32_t &i_len)
- uint32_t **flattenSize** (void)
- void **printStruct** (void)

Public Attributes

- std::string **chipType**

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

- std::string **chipShortType**

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

- std::string **chipCommonType**

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

- uint32_t **pos**

  (Detail: Low) Position of this entry

- uint32_t **unitId**

  (Detail: High) Unit Id of this entry

- uint8_t **numProcCores**

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

- uint32_t **chipEc**

  (Detail: High) EC level of this chip, 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 Constructor & Destructor Documentation

4.4.2.1 ecmdChipData::ecmdChipData () [inline]

4.4.2.2 ecmdChipData::~ecmdChipData () [inline]

4.4.3 Member Function Documentation

4.4.3.1 uint32_t ecmdChipData::flatten (uint8_t * *o_buf*, uint32_t & *i_len*)

4.4.3.2 uint32_t ecmdChipData::unflatten (const uint8_t * *i_buf*, uint32_t & *i_len*)

4.4.3.3 uint32_t ecmdChipData::flattenSize (void)

4.4.3.4 void ecmdChipData::printStruct (void)

4.4.4 Member Data Documentation

4.4.4.1 std::string ecmdChipData::chipType

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

4.4.4.2 std::string ecmdChipData::chipShortType

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

4.4.4.3 std::string ecmdChipData::chipCommonType

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

4.4.4.4 uint32_t ecmdChipData::pos

(Detail: Low) Position of this entry

4.4.4.5 uint32_t ecmdChipData::unitId

(Detail: High) Unit Id of this entry

4.4.4.6 uint8_t ecmdChipData::numProcCores

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

4.4.4.7 uint32_t ecmdChipData::chipEc

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

4.4.4.8 uint32_t ecmdChipData::simModelEc

(Detail: High) Model EC level of this chip

4.4.4.9 ecmdChipInterfaceType_t ecmdChipData::interfaceType

(Detail: High) Interface Macro used by the chip

4.4.4.10 uint32_t ecmdChipData::chipFlags

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

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

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

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

- **ecmdStructs.H**

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 Member Functions

- **ecmdChipTarget** ()
- **~ecmdChipTarget** ()
- uint32_t **flatten** (uint8_t *o_buf, uint32_t i_len)
- uint32_t **unflatten** (const uint8_t *i_buf, uint32_t i_len)
- uint32_t **flattenSize** (void) const
- void **printStruct** (void) const

Public Attributes

- uint32_t **cage**

  cage that contains node with chip

- uint32_t **node**

  node that contains chip

- uint32_t **slot**

  Card Slot/Fru to target.

- std::string **chipType**

  name of chip to access , either actual or common name

- uint32_t **pos**

  position of chip within node

- uint8_t **core**

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

- uint8_t **thread**

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

- uint32_t **unitId**

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

- **ecmdChipTargetState_t cageState**

  cage field state

- **ecmdChipTargetState_t nodeState**

  node field state

- **ecmdChipTargetState_t slotState**

  slot field state

- **ecmdChipTargetState_t chipTypeState**

  chipType field state

- **ecmdChipTargetState_t posState**

  pos field state

- **ecmdChipTargetState_t coreState**

  core field state

- **ecmdChipTargetState_t threadState**

  thread field state

- **ecmdChipTargetState_t unitIdState**

  unitId field state

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 Constructor & Destructor Documentation

4.5.2.1 ecmdChipTarget::ecmdChipTarget () [inline]

4.5.2.2 ecmdChipTarget::~ecmdChipTarget () [inline]

4.5.3 Member Function Documentation

4.5.3.1 uint32_t ecmdChipTarget::flatten (uint8_t * *o_buf*, uint32_t *i_len*)

4.5.3.2 uint32_t ecmdChipTarget::unflatten (const uint8_t * *i_buf*, uint32_t *i_len*)

4.5.3.3 uint32_t ecmdChipTarget::flattenSize (void) const

4.5.3.4 void ecmdChipTarget::printStruct (void) const

4.5.4 Member Data Documentation

4.5.4.1 uint32_t ecmdChipTarget::cage

cage that contains node with chip

4.5.4.2 uint32_t ecmdChipTarget::node

node that contains chip

4.5.4.3 uint32_t ecmdChipTarget::slot

Card Slot/Fru to target.

4.5.4.4 std::string ecmdChipTarget::chipType

name of chip to access , either actual or common name

4.5.4.5 uint32_t ecmdChipTarget::pos

position of chip within node

4.5.4.6 uint8_t ecmdChipTarget::core

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

4.5.4.7 uint8_t ecmdChipTarget::thread

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

4.5.4.8 uint32_t ecmdChipTarget::unitId

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

4.5.4.9 ecmdChipTargetState_t ecmdChipTarget::cageState

cage field state

4.5.4.10 ecmdChipTargetState_t ecmdChipTarget::nodeState

node field state

4.5.4.11 ecmdChipTargetState_t ecmdChipTarget::slotState

slot field state

4.5.4.12 ecmdChipTargetState_t ecmdChipTarget::chipTypeState

chipType field state

4.5.4.13 ecmdChipTargetState_t ecmdChipTarget::posState

pos field state

4.5.4.14 ecmdChipTargetState_t ecmdChipTarget::coreState

core field state

4.5.4.15 ecmdChipTargetState_t ecmdChipTarget::threadState

thread field state

4.5.4.16 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 Member Functions

- **ecmdCoreData** ()
- **~ecmdCoreData** ()
- uint32_t **flatten** (uint8_t *o_buf, uint32_t &i_len)
- uint32_t **unflatten** (const uint8_t *i_buf, uint32_t &i_len)
- uint32_t **flattenSize** (void)
- void **printStruct** (void)

Public Attributes

- uint8_t **coreId**

  (Detail: Low) core number of this entry

- uint8_t **numProcThreads**

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

- uint32_t **unitId**

  (Detail: High) Unit Id of this entry

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

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

4.6.1 Detailed Description

Used for the ecmdQueryConfig function to return core data.

Operators Supported : <

4.6.2 Constructor & Destructor Documentation

4.6.2.1 ecmdCoreData::ecmdCoreData () [inline]

4.6.2.2 ecmdCoreData::~ecmdCoreData () [inline]

4.6.3 Member Function Documentation

4.6.3.1 uint32_t ecmdCoreData::flatten (uint8_t * *o_buf*, uint32_t & *i_len*)

4.6.3.2 uint32_t ecmdCoreData::unflatten (const uint8_t * *i_buf*, uint32_t & *i_len*)

4.6.3.3 uint32_t ecmdCoreData::flattenSize (void)

4.6.3.4 void ecmdCoreData::printStruct (void)

4.6.4 Member Data Documentation

4.6.4.1 uint8_t ecmdCoreData::coreId

(Detail: Low) core number of this entry

4.6.4.2 uint8_t ecmdCoreData::numProcThreads

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

4.6.4.3 uint32_t ecmdCoreData::unitId

(Detail: High) Unit Id of this entry

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

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

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

- **ecmdStructs.H**

4.7 ecmdDataBuffer Class Reference

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

`#include <ecmdDataBuffer.H>`

Public Member Functions

ecmdDataBuffer Constructors

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

Buffer Size Functions

- uint32_t **clear** ()
  Called by the destructor, available to user to reset buffer to default constructor state.
- uint32_t **getWordLength** () const
  Return the length of the buffer in words.
- uint32_t **getByteLength** () const
  Return the length of the buffer in bytes.
- uint32_t **getBitLength** () const
  Return the length of the buffer in bits.
- uint32_t **getCapacity** () const
  Return the actual capacity of the internal buffer in words.
- uint32_t **setWordLength** (uint32_t i_newNumWords)
  Reinitialize the Buffer to specified length.
- uint32_t **setBitLength** (uint32_t i_newNumBits)
  Reinitialize the Buffer to specified length.
- uint32_t **setCapacity** (uint32_t i_newNumWords)
  Reinitialize the internal buffer to specified length.
- uint32_t **shrinkBitLength** (uint32_t i_newNumBits)
  Shrink buffer size to a new bit size.

Bit/Word Manipulation Functions

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

Buffer Manipulation Functions

- uint32_t **shiftRight** (uint32_t i_shiftnum)
  Shift data to right.
- uint32_t **shiftLeft** (uint32_t i_shiftnum)

Shift data to left.

- uint32_t **shiftRightAndResize** (uint32_t i_shiftnum)
  Shift data to right - resizing buffer.

- uint32_t **shiftLeftAndResize** (uint32_t i_shiftnum)
  Shift data to left - resizing buffer.

- uint32_t **rotateRight** (uint32_t i_rotatenum)
  Rotate data to right.

- uint32_t **rotateLeft** (uint32_t i_rotatenum)
  Rotate data to left.

- uint32_t **flushTo0** ()
  Clear entire buffer to 0's.

- uint32_t **flushTo1** ()
  Set entire buffer to 1's.

- uint32_t **invert** ()
  Invert entire buffer.

- uint32_t **reverse** ()
  Bit reverse entire buffer.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

- uint32_t **setAnd** (const uint32_t *i_datain, uint32_t i_startbit, uint32_t i_len)
  AND data into DataBuffer.
- uint32_t **setAnd** (uint32_t i_datain, uint32_t i_startbit, uint32_t i_len)
  AND data into DataBuffer.
- uint32_t **copy** (**ecmdDataBuffer** &o_copyBuffer) const
  Copy entire contents of this ecmdDataBuffer into o_copyBuffer.
- **ecmdDataBuffer** & **operator=** (const **ecmdDataBuffer** &i_master)
  Copy Constructor.
- uint32_t **memCopyIn** (const uint32_t *i_buf, uint32_t i_bytes)
  Copy buffer into this ecmdDataBuffer.
- uint32_t **memCopyOut** (uint32_t *o_buf, uint32_t i_bytes) const
  Copy DataBuffer into supplied uint32_t buffer.
- uint32_t **flatten** (uint8_t *o_data, uint32_t i_len) const
  Flatten all the object data into a uint8_t buffer.
- uint32_t **unflatten** (const uint8_t *i_data, uint32_t i_len)
  Unflatten object data from a uint8_t buffer into this DataBuffer.
- uint32_t **flattenSize** (void) const
  Return number of bytes needed for a buffer to flatten the object.

Parity Functions

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

Buffer Character Conversion Functions

- std::string **genHexLeftStr** (uint32_t i_start, uint32_t i_bitlen) const
  Return Data as a hex left aligned char string.
- std::string **genHexRightStr** (uint32_t i_start, uint32_t i_bitlen) const
  Return Data as a hex right aligned char string.
- std::string **genBinStr** (uint32_t i_start, uint32_t i_bitlen) const
  Return Data as a binary char string.
- std::string **genAsciiStr** (uint32_t i_start, uint32_t i_bitlen) const

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

- std::string **genHexLeftStr** () const
  Return entire buffer as a hex left aligned char string.

- std::string **genHexRightStr** () const
  Return entire buffer as a hex right aligned char string.

- std::string **genBinStr** () const
  Return entire buffer as a binary char string.

- std::string **genAsciiStr** () const
  Return Data as an ASCII char string. If it's out of range, a . is printed.

- std::string **genXstateStr** (uint32_t i_start, uint32_t i_bitlen) const
  Retrieve a section of the Xstate Data.

- std::string **genXstateStr** () const
  Retrieve entire Xstate Data buffer.

String to Data conversion functions

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

- uint32_t **insertFromHexLeftAndResize** (const char *i_hexChars, uint32_t i_start=0, uint32_t i_length=0)
  Convert data from a hex left-aligned string and insert it into this data buffer - and set's buffer length to size of data.

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

- uint32_t **insertFromHexRightAndResize** (const char *i_hexChars, uint32_t i_start=0, uint32_t i_expectedLength=0)
  Convert data from a hex right-aligned string and insert it into this data buffer - and set's buffer length to size of data.

- uint32_t **insertFromBin** (const char *i_binChars, uint32_t i_start=0)
  Convert data from a binary string and insert it into this data buffer.

- uint32_t **insertFromBinAndResize** (const char *i_binChars, uint32_t i_start=0)
  Convert data from a binary string and insert it into this data buffer - and set's buffer length to size of data.

Simulation Buffer Functions

- bool **hasXstate** () const
  Check Entire buffer for any X-state values.

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

Check section of buffer for any X-state values.

- char **getXstate** (uint32_t i_bit) const
  Retrieve an Xstate value from the buffer.
- uint32_t **setXstate** (uint32_t i_bit, char i_value)
  Set an Xstate value in the buffer.
- uint32_t **setXstate** (uint32_t i_bitoffset, const char *i_datastr)
  Set a range of Xstate values in buffer.
- uint32_t **memCopyInXstate** (const char *i_buf, uint32_t i_bytes)
  Copy buffer into the Xstate data of this ecmdDataBuffer.
- uint32_t **memCopyOutXstate** (char *o_buf, uint32_t i_bytes) const
  Copy DataBuffer into supplied char buffer from Xstate data.

Misc Functions

- uint32_t **writeFile** (const char *i_filename, **ecmdFormatType_t** i_format)
  Write buffer out into a file in the format specified.
- uint32_t **readFile** (const char *i_filename, **ecmdFormatType_t** i_format)
  Read data from the file into the buffer.
- uint32_t **shareBuffer** (**ecmdDataBuffer** *i_sharingBuffer)
  This function will take the passed in buffer, delete any current data it holds, and point its data var to that which is owned by the one being called with. It will not have iv_UserOwned flag set, so it should not delete the buffer it points to, nor resize it, but it can alter the data. The use of this function is for caching data for reads.

Operator overloads

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

Protected Member Functions

- uint32_t **fillDataStr** (char fillChar)

Protected Attributes

- uint32_t **iv_Capacity**
  Actual buffer capacity - always >= iv_NumWords.
- uint32_t **iv_NumWords**
  Specified buffer size rounded to next word.
- uint32_t **iv_NumBits**
  Specified buffer size in bits.
- uint32_t * **iv_Data**
  Pointer to buffer inside iv_RealData.
- uint32_t * **iv_RealData**
  Real buffer - with header and tail.
- bool **iv_UserOwned**
  Whether or not this buffer owns the data.
- char * **iv_DataStr**

Friends

- class **ecmdDataBufferImplementationHelper**

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 (uint32_t *i_numBits*)

Constructor.

Parameters:
i_numBits Size of data in bits to initialize

Postcondition:
ecmdDataBuffer is initialized and zero'd out

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

Copy Constructor.

Parameters:
other Buffer to copy

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

Default Destructor.

4.7.3 Member Function Documentation

4.7.3.1 uint32_t ecmdDataBuffer::clear ()

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

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

Postcondition:
Memory deallocated and size set to 0

4.7.3.2 uint32_t ecmdDataBuffer::getWordLength () const

Return the length of the buffer in words.

Return values:
Buffer length in words rounded up

4.7.3.3 uint32_t ecmdDataBuffer::getByteLength () const

Return the length of the buffer in bytes.

Return values:
Buffer length in bytes rounded up

4.7.3.4 uint32_t ecmdDataBuffer::getBitLength () const

Return the length of the buffer in bits.

Return values:
Buffer length in bits

4.7.3.5 uint32_t ecmdDataBuffer::getCapacity () const

Return the actual capacity of the internal buffer in words.

Return values:
Actual capacity in words of internal buffer

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

Reinitialize the Buffer to specified length.

Parameters:
i_newNumWords Length of new buffer in words

Postcondition:
Buffer is reinitialized and zero'd out

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

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

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

Reinitialize the Buffer to specified length.

Parameters:
i_newNumBits Length of new buffer in bits

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

Postcondition:
Buffer is reinitialized and zero'd out

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

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

Reinitialize the internal buffer to specified length.

Parameters:
i_newNumWords length of internal data buffer in words

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

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

CAUTION : All data stored in buffer will be lost

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

Shrink buffer size to a new bit size.

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

Return values:
ECMD_DBUF_SUCCESS on success

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

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

Turn on a bit in buffer.

Parameters:
i_bit Bit in buffer to turn on

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

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

Turn on a bit in buffer.

Parameters:
i_bit start bit in buffer to turn on
i_len Number of consecutive bits from start bit to turn on

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

4.7.3.12 uint32_t ecmdDataBuffer::writeBit (uint32_t *i_ bit*, uint32_t *i_ value*)

Write a bit to specified value in buffer.

Parameters:

i_ bit Bit in buffer to turn on

i_ value Value to write

Return values:

ECMD_ DBUF_ SUCCESS on success

ECMD_ DBUF_ BUFFER_ OVERFLOW i_bit is not contained in the size of this buffer

4.7.3.13 uint32_t ecmdDataBuffer::setWord (uint32_t *i_ wordoffset*, uint32_t *i_ value*)

Set a word of data in buffer.

Parameters:

i_ wordoffset Offset of word to set

i_ value 32 bits of data to put into word

Return values:

ECMD_ DBUF_ SUCCESS on success

ECMD_ DBUF_ BUFFER_ OVERFLOW i_wordoffset is not contained in the size of this buffer

4.7.3.14 uint32_t ecmdDataBuffer::getWord (uint32_t *i_ wordoffset*) const

Fetch a word from ecmdDataBuffer.

Parameters:

i_ wordoffset Offset of word to fetch

Return values:

Value of word requested

4.7.3.15 uint32_t ecmdDataBuffer::setByte (uint32_t *i_ byteoffset*, uint8_t *i_ value*)

Set a byte of data in buffer.

Parameters:

i_ byteoffset Offset of byte to set

i_ value 8 bits of data to put into byte

Return values:

ECMD_ DBUF_ SUCCESS on success

ECMD_ DBUF_ BUFFER_ OVERFLOW i_byteoffset is not contained in the size of this buffer

4.7.3.16 uint8_t ecmdDataBuffer::getByte (uint32_t *i_ byteoffset*) const

Fetch a byte from ecmdDataBuffer.

Parameters:
i_ byteoffset Offset of byte to fetch

Return values:
Value of byte requested

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

4.7.3.17 uint32_t ecmdDataBuffer::clearBit (uint32_t *i_ bit*)

Clear a bit in buffer.

Parameters:
i_ bit Bit in buffer to turn off

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

4.7.3.18 uint32_t ecmdDataBuffer::clearBit (uint32_t *i_ bit*, uint32_t *i_ len*)

Clear multiple bits in buffer.

Parameters:
i_ bit Start bit in buffer to turn off
i_ len Number of consecutive bits from start bit to off

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

4.7.3.19 uint32_t ecmdDataBuffer::flipBit (uint32_t *i_ bit*)

Invert bit.

Parameters:
i_ bit Bit in buffer to invert

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

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

Invert multiple bits.

Parameters:

i_bit Start bit in buffer to invert

i_len Number of consecutive bits to invert

Return values:

ECMD_DBUF_SUCCESS on success

ECMD_DBUF_BUFFER_OVERFLOW i_bit is not contained in the size of this buffer

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

Test if bit is set.

Parameters:

i_bit Bit to test

Return values:

true if bit is set - false if bit is clear

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

Test if multiple bits are set.

Parameters:

i_bit Start bit to test

i_len Number of consecutive bits to test

Return values:

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

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

Test if bit is clear.

Parameters:

i_bit Bit to test

Return values:

true if bit is clear - false if bit is set

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

Test if multiple bits are clear.

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

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

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

Count number of bits set in a range.

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

Return values:
Number of bits set in range

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

Shift data to right.

Parameters:
i_shiftnum Number of bits to shift

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

Return values:
ECMD_DBUF_SUCCESS on success

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

Shift data to left.

Parameters:
i_shiftnum Number of bits to shift

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

Return values:
ECMD_DBUF_SUCCESS on success

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

Shift data to right - resizing buffer.

Parameters:
i_shiftnum Number of bits to shift

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

Return values:
ECMD_DBUF_SUCCESS on success

ECMD_DBUF_NOT_OWNER when called on buffer not owned

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

Shift data to left - resizing buffer.

Parameters:
i_shiftnum Number of bits to shift

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

Return values:
ECMD_DBUF_SUCCESS on success

ECMD_DBUF_NOT_OWNER when called on buffer not owned

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

Rotate data to right.

Parameters:
i_rotatenum Number of bits to rotate

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

Return values:
ECMD_DBUF_SUCCESS on success

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

Rotate data to left.

Parameters:
i_rotatenum Number of bits to rotate

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

Return values:
ECMD_DBUF_SUCCESS on success

4.7.3.32 uint32_t ecmdDataBuffer::flushTo0 ()

Clear entire buffer to 0's.

Return values:
ECMD_DBUF_SUCCESS on success

4.7.3.33 uint32_t ecmdDataBuffer::flushTo1 ()

Set entire buffer to 1's.

Return values:
ECMD_DBUF_SUCCESS on success

4.7.3.34 uint32_t ecmdDataBuffer::invert ()

Invert entire buffer.

Return values:
ECMD_DBUF_SUCCESS on success

4.7.3.35 uint32_t ecmdDataBuffer::reverse ()

Bit reverse entire buffer.

Return values:
ECMD_DBUF_SUCCESS on success

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

Apply an inversion mask to data inside buffer.

Parameters:

i_ invMask Buffer that stores inversion mask

i_ invByteLen Buffer length provided in bytes

Return values:

ECMD_ DBUF_ SUCCESS on success

4.7.3.37 uint32_t ecmdDataBuffer::applyInversionMask (const ecmdDataBuffer & *i_ invMaskBuffer*, uint32_t *i_ invByteLen*)

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

Parameters:

i_ invMaskBuffer Buffer that stores inversion mask

i_ invByteLen Buffer length provided in bytes

Return values:

ECMD_ DBUF_ SUCCESS on success

4.7.3.38 uint32_t ecmdDataBuffer::insert (const ecmdDataBuffer & *i_ bufferIn*, uint32_t *i_ targetStart*, uint32_t *i_ len*, uint32_t *i_ sourceStart* = 0)

Copy part of another DataBuffer into this one.

Parameters:

i_ bufferIn DataBuffer to copy data from - data is taken left aligned

i_ targetStart Start bit to insert to

i_ len Length of bits to insert

i_ sourceStart Start bit in i_bufferIn - default value is zero

Precondition:

DataBuffer must be pre-allocated

Postcondition:

Data is copied from i_bufferIn to this DataBuffer in specified location

Return values:

ECMD_ DBUF_ SUCCESS on success

ECMD_ DBUF_ BUFFER_ OVERFLOW operation requested out of range

4.7.3.39 uint32_t ecmdDataBuffer::insert (const uint32_t * *i_ datain*, uint32_t *i_ targetStart*, uint32_t *i_ len*, uint32_t *i_ sourceStart* = 0)

Copy part of a uint32_t array into this DataBuffer.

Parameters:

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

i_ targetStart Start bit to insert into

i_ len Length of bits to insert

i_ sourceStart Start bit in i_datain - default value is zero

Precondition:
DataBuffer must be pre-allocated

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

Return values:
ECMD_ DBUF_ SUCCESS on success

ECMD_ DBUF_ BUFFER_ OVERFLOW operation requested out of range

4.7.3.40 uint32_t ecmdDataBuffer::insert (uint32_t *i_ datain*, uint32_t *i_ targetStart*, uint32_t *i_ len*, uint32_t *i_ sourceStart* = 0)

Copy part of a uint32_t into the DataBuffer.

Parameters:
i_ datain uint32_t value to copy into DataBuffer - data is taken left aligned

i_ targetStart Start bit to insert into

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

i_ sourceStart Start bit in i_datain - default value is zero

Precondition:
DataBuffer must be pre-allocated

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

Return values:
ECMD_ DBUF_ SUCCESS on success

ECMD_ DBUF_ BUFFER_ OVERFLOW operation requested out of range

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

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

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

i_ start Start bit to insert into

i_ len Length of bits to insert

Precondition:
DataBuffer must be pre-allocated

Postcondition:
Data is copied from datain into this DataBuffer at specified location

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

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

4.7.3.42 uint32_t ecmdDataBuffer::insertFromRight (uint32_t *i_datain*, uint32_t *i_start*, uint32_t *i_len*)

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

Parameters:
i_datain uint32_t value to copy into DataBuffer - data is taken right aligned
i_start Start bit to insert into
i_len Length of bits to insert (must be <= 32)

Precondition:
DataBuffer must be pre-allocated

Postcondition:
Data is copied from datain into this DataBuffer at specified location

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

4.7.3.43 uint32_t ecmdDataBuffer::extract (ecmdDataBuffer & *o_bufferOut*, uint32_t *i_start*, uint32_t *i_len*) const

Copy data from this DataBuffer into another.

Parameters:
o_bufferOut DataBuffer to copy into - data is placed left aligned
i_start Start bit of data in this DataBuffer to copy
i_len Length of consecutive bits to copy

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

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

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

4.7.3.44 uint32_t ecmdDataBuffer::extract (uint32_t * *o_data*, uint32_t *i_start*, uint32_t *i_len*) const

Copy data from this DataBuffer into another.

Parameters:

o_data uint32_t buffer to copy into - data is placed left aligned - must be pre-allocated

i_start Start bit of data in DataBuffer to copy

i_len Length of consecutive bits to copy

Postcondition:

Data is copied from specified location in this DataBuffer to o_data

Return values:

ECMD_DBUF_SUCCESS on success

ECMD_DBUF_BUFFER_OVERFLOW operation requested out of range

4.7.3.45 uint32_t ecmdDataBuffer::extractPreserve (ecmdDataBuffer & *o_bufferOut*, uint32_t *i_start*, uint32_t *i_len*, uint32_t *i_targetStart* = 0) const

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

Parameters:

o_bufferOut Target data buffer where data is copied into

i_start Start bit in this DataBuffer to begin copy

i_len Length of consecutive bits to copy

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

Postcondition:

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

Return values:

ECMD_DBUF_SUCCESS on success

EMCD_DBUF_BUFFER_OVERFLOW data requested is out of range in one of the 2 buffers

4.7.3.46 uint32_t ecmdDataBuffer::extractPreserve (uint32_t * *o_data*, uint32_t *i_start*, uint32_t *i_len*, uint32_t *i_targetStart* = 0) const

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

Parameters:

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

i_start Start bit in this DataBuffer to begin the copy

i_len Length of consecutive bits to copy

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

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

Return values:
ECMD_ DBUF_ SUCCESS on success

ECMD_ DBUF_ INIT_ FAIL unable to allocate databuffer

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

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

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

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

i_ start Start bit of data in DataBuffer to copy

i_ len Length of consecutive bits to copy

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

Return values:
ECMD_ DBUF_ SUCCESS on success

ECMD_ DBUF_ BUFFER_ OVERFLOW operation requested out of range

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

Copy data from this DataBuffer into a uint32_t buffer.

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

i_ start Start bit of data in DataBuffer to copy

i_ len Length of consecutive bits to copy

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

Return values:
ECMD_ DBUF_ SUCCESS on success

ECMD_ DBUF_ BUFFER_ OVERFLOW operation requested out of range

4.7.3.49 uint32_t ecmdDataBuffer::concat (const ecmdDataBuffer & *i_ buf0*, const ecmdDataBuffer & *i_ buf1*)

Concatenate 2 DataBuffers into in this one.

Parameters:

i_ buf0 First DataBuffer to concatenate; copied to beginning of this buffer

i_ buf1 Second DataBuffer to concatenate; copied to this buffer after the first buffer

Postcondition:

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

Return values:

ECMD_DBUF_SUCCESS on success

ECMD_DBUF_BUFFER_OVERFLOW operation requested out of range

4.7.3.50 uint32_t ecmdDataBuffer::concat (const ecmdDataBuffer & *i_ buf0*, const ecmdDataBuffer & *i_ buf1*, const ecmdDataBuffer & *i_ buf2*)

Concatenate 3 DataBuffers into in this one.

Parameters:

i_ buf0 First DataBuffer to concatenate; copied to beginning of this buffer

i_ buf1 Second DataBuffer to concatenate; copied to this buffer after the first buffer

i_ buf2 Third DataBuffer to concatenate; copied to this buffer after the second buffer

Postcondition:

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

Return values:

ECMD_DBUF_SUCCESS on success

ECMD_DBUF_BUFFER_OVERFLOW operation requested out of range

4.7.3.51 uint32_t ecmdDataBuffer::setOr (const ecmdDataBuffer & *i_ bufferIn*, uint32_t *i_ startbit*, uint32_t *i_ len*)

OR data into DataBuffer.

Parameters:

i_ bufferIn DataBuffer to OR data from - data is taken left aligned

i_ startbit Start bit to OR to

i_ len Length of bits to OR

Postcondition:

Data is ORed from i_bufferIn to this DataBuffer in specified location

Return values:

ECMD_DBUF_SUCCESS on success

ECMD_DBUF_BUFFER_OVERFLOW operation requested out of range

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

OR data into DataBuffer.

Parameters:

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

i_ startbit Start bit to OR to

i_ len Length of bits to OR

Postcondition:

Data is ORed from datain to this DataBuffer in specified location

Return values:

ECMD_DBUF_SUCCESS on success

ECMD_DBUF_BUFFER_OVERFLOW operation requested out of range

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

OR data into DataBuffer.

Parameters:

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

i_ startbit Start bit to OR to

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

Postcondition:

Data is ORed from datain to this DataBuffer in specified location

Return values:

ECMD_DBUF_SUCCESS on success

ECMD_DBUF_BUFFER_OVERFLOW operation requested out of range

4.7.3.54 uint32_t ecmdDataBuffer::merge (const ecmdDataBuffer & *i_ bufferIn*)

OR data into DataBuffer.

Parameters:

i_ bufferIn DataBuffer to OR data from - data is taken left aligned

Postcondition:

Entire data is ORed from bufferIn to this DataBuffer

Return values:

ECMD_DBUF_SUCCESS on success

ECMD_DBUF_BUFFER_OVERFLOW operation requested out of range

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

XOR data into DataBuffer.

Parameters:

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

i_startbit Start bit to XOR to

i_len Length of bits to XOR

Postcondition:

Data is XORed from i_bufferIn to this DataBuffer in specified location

Return values:

ECMD_DBUF_SUCCESS on success

ECMD_DBUF_BUFFER_OVERFLOW operation requested out of range

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

XOR data into DataBuffer.

Parameters:

i_datain uint32_t buffer to XOR data from - data is taken left aligned

i_startbit Start bit to XOR to

i_len Length of bits to XOR

Postcondition:

Data is XORed from datain to this DataBuffer in specified location

Return values:

ECMD_DBUF_SUCCESS on success

ECMD_DBUF_BUFFER_OVERFLOW operation requested out of range

4.7.3.57 uint32_t ecmdDataBuffer::setXor (uint32_t *i_datain*, uint32_t *i_startbit*, uint32_t *i_len*)

XOR data into DataBuffer.

Parameters:

i_datain uint32_t to XOR data from - data is taken left aligned

i_startbit Start bit to XOR to

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

Postcondition:

Data is XORed from datain to this DataBuffer in specified location

Return values:

ECMD_DBUF_SUCCESS on success

ECMD_DBUF_BUFFER_OVERFLOW operation requested out of range

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

AND data into DataBuffer.

Parameters:

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

i_ startbit Start bit to AND to

i_ len Length of bits to AND

Postcondition:

Data is ANDed from bufferIn to this DataBuffer in specified location

Return values:

ECMD_DBUF_SUCCESS on success

ECMD_DBUF_BUFFER_OVERFLOW operation requested out of range

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

AND data into DataBuffer.

Parameters:

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

i_ startbit Start bit to AND to

i_ len Length of bits to AND

Postcondition:

Data is ANDed from datain to this DataBuffer in specified location

Return values:

ECMD_DBUF_SUCCESS on success

ECMD_DBUF_BUFFER_OVERFLOW operation requested out of range

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

AND data into DataBuffer.

Parameters:

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

i_ startbit Start bit to AND to

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

Postcondition:

Data is ANDed from datain to this DataBuffer in specified location

Return values:

ECMD_DBUF_SUCCESS on success

ECMD_DBUF_BUFFER_OVERFLOW operation requested out of range

4.7.3.61 uint32_t ecmdDataBuffer::copy (ecmdDataBuffer & *o_copyBuffer*) const

Copy entire contents of this ecmdDataBuffer into o_copyBuffer.

Parameters:
o_copyBuffer DataBuffer to copy data into

Postcondition:
copyBuffer is allocated, is an exact duplicate of this DataBuffer

Return values:
ECMD_DBUF_SUCCESS on success

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

Copy Constructor.

Parameters:
i_master DataBuffer to copy from

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

4.7.3.63 uint32_t ecmdDataBuffer::memCopyIn (const uint32_t * *i_buf*, uint32_t *i_bytes*)

Copy buffer into this ecmdDataBuffer.

Parameters:
i_buf Buffer to copy from

i_bytes Byte length to copy

Precondition:
DataBuffer must be pre-allocated

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

Return values:
ECMD_DBUF_SUCCESS on success

ECMD_DBUF_BUFFER_OVERFLOW operation requested out of range

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

Copy DataBuffer into supplied uint32_t buffer.

Parameters:

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

i_ bytes Byte length to copy

Postcondition:

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

Return values:

ECMD_DBUF_SUCCESS on success

ECMD_DBUF_BUFFER_OVERFLOW operation requested out of range

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

Flatten all the object data into a uint8_t buffer.

Parameters:

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

i_ len Number of bytes in the o_data buffer

Postcondition:

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

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

Unflatten object data from a uint8_t buffer into this DataBuffer.

Parameters:

i_ data Byte buffer to read the flattened data from

i_ len Number of bytes in the i_data buffer

Postcondition:

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

4.7.3.67 uint32_t ecmdDataBuffer::flattenSize (void) const

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

Return values:

Number of bytes needed

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

Generate odd parity over a range of bits.

Parameters:

i_start Start bit of range

i_stop Stop bit of range

Return values:

0 or 1 depending on parity of range

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

Generate even parity over a range of bits.

Parameters:

i_start Start bit of range

i_stop Stop bit of range

Return values:

0 or 1 depending on parity of range

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

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

Parameters:

i_start Start bit of range

i_stop Stop bit of range

i_insertpos Bit position to insert parity

Return values:

ECMD_DBUF_SUCCESS on success

ECMD_DBUF_BUFFER_OVERFLOW operation requested out of range

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

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

Parameters:

i_start Start bit of range

i_stop Stop bit of range

i_ insertpos Bit position to insert parity

Return values:

ECMD_ DBUF_ SUCCESS on success

ECMD_ DBUF_ BUFFER_ OVERFLOW operation requested out of range

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

Return Data as a hex left aligned char string.

Parameters:

i_ start Start bit of data to convert

i_ bitlen Number of consecutive bits to convert

Return values:

String containing requested data

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

Return Data as a hex right aligned char string.

Parameters:

i_ start Start bit of data to convert

i_ bitlen Number of consecutive bits to convert

Return values:

String containing requested data

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

Return Data as a binary char string.

Parameters:

i_ start Start bit of data to convert

i_ bitlen Number of consecutive bits to convert

Return values:

String containing requested data

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

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

Parameters:

i_start Start bit of data to convert

i_bitlen Number of consecutive bits to convert

Return values:

String containing requested data

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

Return entire buffer as a hex left aligned char string.

Return values:

String containing requested data

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

Return entire buffer as a hex right aligned char string.

Return values:

String containing requested data

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

Return entire buffer as a binary char string.

Return values:

String containing requested data

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

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

Return values:

String containing requested data

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

Retrieve a section of the Xstate Data.

Parameters:

i_start Start bit of data to retrieve

i_ bitlen Number of consecutive bits to retrieve

Return values:

String containing requested data

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

Retrieve entire Xstate Data buffer.

Return values:

String containing requested data

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

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

Parameters:

i_ hexChars Hex Left-aligned string of data to insert

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

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

Return values:

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

ECMD_ SUCCESS on success

non-zero on failure

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

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

Parameters:

i_ hexChars Hex Left-aligned string of data to insert

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

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

Return values:

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

ECMD_ SUCCESS on success

non-zero on failure

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

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

Parameters:

i_ hexChars Hex Right-aligned string of data to insert

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

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

Return values:

ECMD_DBUF_INVALID_DATA_FORMAT if non-hex chars detected in i_hexChars

ECMD_SUCCESS on success

non-zero on failure

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

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

Parameters:

i_ hexChars Hex Right-aligned string of data to insert

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

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

Return values:

ECMD_DBUF_INVALID_DATA_FORMAT if non-hex chars detected in i_hexChars

ECMD_SUCCESS on success

non-zero on failure

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

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

Return values:

0 on success- non-zero on failure

Parameters:

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

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

Return values:

ECMD_DBUF_INVALID_DATA_FORMAT if non-binary chars detected in i_binChars

ECMD_SUCCESS on success

non-zero on failure

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

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

Return values:

0 on success- non-zero on failure

Parameters:

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

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

Return values:

ECMD_DBUF_INVALID_DATA_FORMAT if non-binary chars detected in i_binChars

ECMD_SUCCESS on success

non-zero on failure

4.7.3.88 bool ecmdDataBuffer::hasXstate () const

Check Entire buffer for any X-state values.

Return values:

1 if xstate found 0 if none

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

Check section of buffer for any X-state values.

Parameters:

i_start Start bit to test

i_length Number of consecutive bits to test

Return values:

1 if xstate found 0 if none

4.7.3.90 char ecmdDataBuffer::getXstate (uint32_t *i_ bit*) const

Retrieve an Xstate value from the buffer.

Parameters:
i_ bit Bit to retrieve

NOTE - To retrieve multiple bits use genXstateStr

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

Set an Xstate value in the buffer.

Parameters:
i_ bit Bit to set
i_ value Xstate value to set

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

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

Set a range of Xstate values in buffer.

Parameters:
i_ bitoffset bit in buffer to start inserting
i_ datastr Character value to set bit - can be "0XX0", "1", "X"

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

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

Copy buffer into the Xstate data of this ecmdDataBuffer.

Parameters:
i_ buf Buffer to copy from
i_ bytes Byte length to copy (char length)

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

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

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

Copy DataBuffer into supplied char buffer from Xstate data.

Parameters:

o_buf Buffer to copy into - must be pre-allocated

i_bytes Byte length to copy (char length)

Postcondition:

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

Return values:

ECMD_DBUF_SUCCESS on success

ECMD_DBUF_BUFFER_OVERFLOW operation requested out of range

4.7.3.95 uint32_t ecmdDataBuffer::writeFile (const char * *i_filename*, ecmdFormatType_t *i_format*)

Write buffer out into a file in the format specified.

Parameters:

i_filename file to write to

i_format format to write in

Return values:

ECMD_DBUF_SUCCESS on success

ECMD_DBUF_FOPEN_FAIL Unable to open the file for write

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

4.7.3.96 uint32_t ecmdDataBuffer::readFile (const char * *i_filename*, ecmdFormatType_t *i_format*)

Read data from the file into the buffer.

Parameters:

i_filename to read from

i_format data format to expect in the file

Return values:

ECMD_DBUF_SUCCESS on success

ECMD_DBUF_FILE_FORMAT_MISMATCH specified format not found in the file

ECMD_DBUF_FOPEN_FAIL Unable to open the file for read

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

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

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

Parameters:
i_sharingBuffer input buffer

Return values:
ECMD_DBUF_SUCCESS on success

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

Overload the == operator.

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

Overload the != operator.

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

Overload the & operator.

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

Overload the | operator.

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

4.7.4 Friends And Related Function Documentation

4.7.4.1 friend class ecmdDataBufferImplementationHelper [friend]

4.7.5 Member Data Documentation

4.7.5.1 uint32_t ecmdDataBuffer::iv_Capacity [protected]

Actual buffer capacity - always >= iv_NumWords.

4.7.5.2 uint32_t ecmdDataBuffer::iv_NumWords [protected]

Specified buffer size rounded to next word.

4.7.5.3 uint32_t ecmdDataBuffer::iv_NumBits [protected]

Specified buffer size in bits.

4.7.5.4 uint32_t* ecmdDataBuffer::iv_Data [protected]

Pointer to buffer inside iv_RealData.

4.7.5.5 uint32_t* ecmdDataBuffer::iv_RealData [protected]

Real buffer - with header and tail.

4.7.5.6 bool ecmdDataBuffer::iv_UserOwned [protected]

Whether or not this buffer owns the data.

4.7.5.7 char* ecmdDataBuffer::iv_DataStr [protected]

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

- **ecmdDataBuffer.H**

4.8 ecmdDataBufferImplementationHelper Class Reference

This is used to help low-level implementation of the **ecmdDataBuffer**(p. 25), this CAN NOT be used by any eCMD client or data corruption will occur.

`#include <ecmdDataBuffer.H>`

Static Public Member Functions

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

4.8.1 Detailed Description

This is used to help low-level implementation of the **ecmdDataBuffer**(p. 25), this CAN NOT be used by any eCMD client or data corruption will occur.

4.8.2 Member Function Documentation

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

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

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

- **ecmdDataBuffer.H**

4.9 ecmdDllInfo Struct Reference

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

`#include <ecmdStructs.H>`

Public Attributes

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

4.9.1 Detailed Description

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

4.9.2 Member Data Documentation

4.9.2.1 ecmdDllType_t ecmdDllInfo::dllType

Dll instance type running.

4.9.2.2 ecmdDllProduct_t ecmdDllInfo::dllProduct

Dll product supported.

4.9.2.3 std::string ecmdDllInfo::dllProductType

Dll product type currently configured.

4.9.2.4 ecmdDllEnv_t ecmdDllInfo::dllEnv

Dll environment (Simulation vs Hardware).

4.9.2.5 std::string ecmdDllInfo::dllBuildDate

Date the Dll was built.

4.9.2.6 std::string ecmdDllInfo::dllCapiVersion

should be set to ECMD_CAPI_VERSION

4.9.2.7 std::string ecmdDllInfo::dllBuildInfo

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

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

- **ecmdStructs.H**

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

- bool **ecmdLooperInitFlag**

  Is fresh ?

- bool **ecmdUseUnitid**

  This looper is looping on unitid targets not config data.

- **ecmdQueryData ecmdSystemConfigData**

  Config data queried from the system.

- std::list< **ecmdCageData** >::iterator **ecmdCurCage**

  Pointer to current Cage.

- std::list< **ecmdNodeData** >::iterator **ecmdCurNode**

  Pointer to current Node.

- std::list< **ecmdSlotData** >::iterator **ecmdCurSlot**

  Pointer to current Slot.

- std::list< **ecmdChipData** >::iterator **ecmdCurChip**

  Pointer to current Chip.

- std::list< **ecmdCoreData** >::iterator **ecmdCurCore**

  Pointer to current Core.

- std::list< **ecmdThreadData** >::iterator **ecmdCurThread**

  Pointer to current Thread.

- **ecmdChipTarget prevTarget**

  Pointer to previous target.

- std::list< **ecmdChipTarget** > **unitIdTargets**

  List of targets if looping on a unitid.

- std::list< **ecmdChipTarget** >::iterator **curUnitIdTarget**

  Pointer to current unitid target.

4.12.1 Detailed Description

Used internally by ecmdConfigLooper to store looping state information.

4.12.2 Member Data Documentation

4.12.2.1 bool ecmdLooperData::ecmdLooperInitFlag

Is fresh ?

4.12.2.2 bool ecmdLooperData::ecmdUseUnitid

This looper is looping on unitid targets not config data.

4.12.2.3 ecmdQueryData ecmdLooperData::ecmdSystemConfigData

Config data queried from the system.

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

Pointer to current Cage.

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

Pointer to current Node.

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

Pointer to current Slot.

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

Pointer to current Chip.

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

Pointer to current Core.

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

Pointer to current Thread.

4.12.2.10 ecmdChipTarget ecmdLooperData::prevTarget

Pointer to previous target.

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

List of targets if looping on a unitid.

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

Pointer to current unitid target.

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 Member Functions

- **ecmdNodeData** ()
- **~ecmdNodeData** ()
- uint32_t **flatten** (uint8_t *o_buf, uint32_t &i_len)
- uint32_t **unflatten** (const uint8_t *i_buf, uint32_t &i_len)
- uint32_t **flattenSize** (void)
- void **printStruct** (void)

Public Attributes

- uint32_t **nodeId**

  (Detail: Low) Node number of this entry

- uint32_t **unitId**

  (Detail: High) Unit Id of this entry

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

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

4.15.1 Detailed Description

Used for the ecmdQueryConfig function to return node data.

Operators Supported : <

4.15.2 Constructor & Destructor Documentation

4.15.2.1 ecmdNodeData::ecmdNodeData () [inline]

4.15.2.2 ecmdNodeData::~ecmdNodeData () [inline]

4.15.3 Member Function Documentation

4.15.3.1 uint32_t ecmdNodeData::flatten (uint8_t * *o_buf*, uint32_t & *i_len*)

4.15.3.2 uint32_t ecmdNodeData::unflatten (const uint8_t * *i_buf*, uint32_t & *i_len*)

4.15.3.3 uint32_t ecmdNodeData::flattenSize (void)

4.15.3.4 void ecmdNodeData::printStruct (void)

4.15.4 Member Data Documentation

4.15.4.1 uint32_t ecmdNodeData::nodeId

(Detail: Low) Node number of this entry

4.15.4.2 uint32_t ecmdNodeData::unitId

(Detail: High) Unit Id of this entry

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

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

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

- **ecmdStructs.H**

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.
- bool **threadReplicated**
  Register is replicated for each thread.

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.

4.16.2.3 bool ecmdProcRegisterInfo::threadReplicated

Register is replicated for each thread.

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 Member Functions

- **ecmdQueryData** ()
- **~ecmdQueryData** ()
- uint32_t **flatten** (uint8_t *o_buf, uint32_t &i_len)
- uint32_t **unflatten** (const uint8_t *i_buf, uint32_t &i_len)
- uint32_t **flattenSize** (void)
- void **printStruct** (void)

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 Constructor & Destructor Documentation

4.17.2.1 ecmdQueryData::ecmdQueryData () [inline]

4.17.2.2 ecmdQueryData::~ecmdQueryData () [inline]

4.17.3 Member Function Documentation

4.17.3.1 uint32_t ecmdQueryData::flatten (uint8_t * *o_buf*, uint32_t & *i_len*)

4.17.3.2 uint32_t ecmdQueryData::unflatten (const uint8_t * *i_buf*, uint32_t & *i_len*)

4.17.3.3 uint32_t ecmdQueryData::flattenSize (void)

4.17.3.4 void ecmdQueryData::printStruct (void)

4.17.4 Member Data Documentation

4.17.4.1 ecmdQueryDetail_t ecmdQueryData::detailLevel

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

4.17.4.2 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.

- std::string **clockDomain**

  Clock domain this ring belongs to.

- **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 std::string ecmdRingData::clockDomain

Clock domain this ring belongs to.

4.18.2.8 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 Member Functions

- **ecmdSlotData** ()
- **~ecmdSlotData** ()
- uint32_t **flatten** (uint8_t *o_buf, uint32_t &i_len)
- uint32_t **unflatten** (const uint8_t *i_buf, uint32_t &i_len)
- uint32_t **flattenSize** (void)
- void **printStruct** (void)

Public Attributes

- uint32_t **slotId**

  (Detail: Low) Slot number of this entry

- uint32_t **unitId**

  (Detail: High) Unit Id of this entry

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

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

4.19.1 Detailed Description

Used for the ecmdQueryConfig function to return slot data.

Operators Supported : <

4.19.2 Constructor & Destructor Documentation

4.19.2.1 ecmdSlotData::ecmdSlotData () [inline]

4.19.2.2 ecmdSlotData::~ecmdSlotData () [inline]

4.19.3 Member Function Documentation

4.19.3.1 uint32_t ecmdSlotData::flatten (uint8_t * *o_buf*, uint32_t & *i_len*)

4.19.3.2 uint32_t ecmdSlotData::unflatten (const uint8_t * *i_buf*, uint32_t & *i_len*)

4.19.3.3 uint32_t ecmdSlotData::flattenSize (void)

4.19.3.4 void ecmdSlotData::printStruct (void)

4.19.4 Member Data Documentation

4.19.4.1 uint32_t ecmdSlotData::slotId

(Detail: Low) Slot number of this entry

4.19.4.2 uint32_t ecmdSlotData::unitId

(Detail: High) Unit Id of this entry

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

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

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

- **ecmdStructs.H**

4.20 ecmdSpyData Struct Reference

Used for the ecmdQuerySpy function to return spy info.

`#include <ecmdStructs.H>`

Public Member Functions

- **ecmdSpyData** ()
- **~ecmdSpyData** ()
- uint32_t **flatten** (uint8_t *o_buf, uint32_t &i_len)
- uint32_t **unflatten** (const uint8_t *i_buf, uint32_t &i_len)
- uint32_t **flattenSize** (void)
- void **printStruct** (void)

Public Attributes

- std::string **spyName**

  Names used to reference this spy.

- int **bitLength**

  length of spy

- **ecmdSpyType_t spyType**

  Type of spy.

- bool **isEccChecked**

  This spy affects some ECC groupings.

- bool **isEnumerated**

  This spy has enumerated values.

- bool **isCoreRelated**

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

- std::string **clockDomain**

  Clock domain this spy belongs to.

- **ecmdClockState_t clockState**

  Required clock state to access this spy.

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

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

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

  Possible epChecker names affected by this Spy.

4.20.1 Detailed Description

Used for the ecmdQuerySpy function to return spy info.

4.20.2 Constructor & Destructor Documentation

4.20.2.1 ecmdSpyData::ecmdSpyData () [inline]

4.20.2.2 ecmdSpyData::~ecmdSpyData () [inline]

4.20.3 Member Function Documentation

4.20.3.1 uint32_t ecmdSpyData::flatten (uint8_t * *o_buf*, uint32_t & *i_len*)

4.20.3.2 uint32_t ecmdSpyData::unflatten (const uint8_t * *i_buf*, uint32_t & *i_len*)

4.20.3.3 uint32_t ecmdSpyData::flattenSize (void)

4.20.3.4 void ecmdSpyData::printStruct (void)

4.20.4 Member Data Documentation

4.20.4.1 std::string ecmdSpyData::spyName

Names used to reference this spy.

4.20.4.2 int ecmdSpyData::bitLength

length of spy

4.20.4.3 ecmdSpyType_t ecmdSpyData::spyType

Type of spy.

4.20.4.4 bool ecmdSpyData::isEccChecked

This spy affects some ECC groupings.

4.20.4.5 bool ecmdSpyData::isEnumerated

This spy has enumerated values.

4.20.4.6 bool ecmdSpyData::isCoreRelated

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

4.20.4.7 std::string ecmdSpyData::clockDomain

Clock domain this spy belongs to.

4.20.4.8 ecmdClockState_t ecmdSpyData::clockState

Required clock state to access this spy.

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

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

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

Possible epChecker names affected by this Spy.

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

- **ecmdStructs.H**

4.21 ecmdSpyGroupData Struct Reference

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

`#include <ecmdStructs.H>`

Public Attributes

- **ecmdDataBuffer extractBuffer**
  The data read from the ring buffer.
- **ecmdDataBuffer deadbitsMask**
  A mask of the bits that were deadbits in that buffer.

4.21.1 Detailed Description

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

4.21.2 Member Data Documentation

4.21.2.1 ecmdDataBuffer ecmdSpyGroupData::extractBuffer

The data read from the ring buffer.

4.21.2.2 ecmdDataBuffer ecmdSpyGroupData::deadbitsMask

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

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

- **ecmdStructs.H**

4.22 ecmdThreadData Struct Reference

Used for the ecmdQueryConfig function to return thread data.

`#include <ecmdStructs.H>`

Public Member Functions

- **ecmdThreadData** ()
- **~ecmdThreadData** ()
- uint32_t **flatten** (uint8_t *o_buf, uint32_t &i_len)
- uint32_t **unflatten** (const uint8_t *i_buf, uint32_t &i_len)
- uint32_t **flattenSize** (void)
- void **printStruct** (void)

Public Attributes

- uint8_t **threadId**

  (Detail: Low) Thread number of this entry

- uint32_t **unitId**

  (Detail: High) Unit Id of this entry

4.22.1 Detailed Description

Used for the ecmdQueryConfig function to return thread data.

Operators Supported : <

4.22.2 Constructor & Destructor Documentation

4.22.2.1 ecmdThreadData::ecmdThreadData () [inline]

4.22.2.2 ecmdThreadData::~ecmdThreadData () [inline]

4.22.3 Member Function Documentation

4.22.3.1 uint32_t ecmdThreadData::flatten (uint8_t * *o_buf*, uint32_t & *i_len*)

4.22.3.2 uint32_t ecmdThreadData::unflatten (const uint8_t * *i_buf*, uint32_t & *i_len*)

4.22.3.3 uint32_t ecmdThreadData::flattenSize (void)

4.22.3.4 void ecmdThreadData::printStruct (void)

4.22.4 Member Data Documentation

4.22.4.1 uint8_t ecmdThreadData::threadId

(Detail: Low) Thread number of this entry

4.22.4.2 uint32_t ecmdThreadData::unitId

(Detail: High) Unit Id of this entry

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

- **ecmdStructs.H**

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>
```

Load/Unload Functions

- uint32_t **cipInitExtension** ()
  Initialize eCMD CIP Extension DLL.

Processor Functions

- uint32_t **cipStartInstructions** (**ecmdChipTarget** &i_target)
  Start Instructions.
- uint32_t **cipStartAllInstructions** ()
  Start Instructions on all configured processors.
- uint32_t **cipStopInstructions** (**ecmdChipTarget** &i_target)
  Stop Instructions.
- uint32_t **cipStopAllInstructions** ()
  Stop All Instructions.
- uint32_t **cipStepInstructions** (**ecmdChipTarget** &i_target, uint32_t i_steps)
  Step Instructions.

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

  Set a hardware breakpoint in Processor using a real address.

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

  Clear a hardware breakpoint from Processor using a real address.

- uint32_t **cipGetVpr** (**ecmdChipTarget** &i_target, uint32_t i_vprNum, **ecmdDataBuffer** &o_data)

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

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

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

- uint32_t **cipPutVpr** (**ecmdChipTarget** &i_target, uint32_t i_vprNum, **ecmdDataBuffer** &i_data)

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

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

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

Memory Functions

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

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

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

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

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

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

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

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

5.1.1 Detailed Description

Cronus & IP eCMD Extension.

Extension Owner : Chris Engel

5.1.2 Function Documentation

5.1.2.1 uint32_t cipInitExtension ()

Initialize eCMD CIP Extension DLL.

Return values:

ECMD_SUCCESS if successful load

ECMD_INVALID_DLL_VERSION if Dll version loaded doesn't match client version

nonzero if unsuccessful

Postcondition:

eCMD CIP Extension is initialized and version checked

5.1.2.2 uint32_t cipStartInstructions (ecmdChipTarget & *i_target*)

Start Instructions.

Parameters:

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

Return values:

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

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

nonzero if unsuccessful

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

5.1.2.3 uint32_t cipStartAllInstructions ()

Start Instructions on all configured processors.

Return values:

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

ECMD_SUCCESS if successful

nonzero if unsuccessful

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

5.1.2.4 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

5.1.2.5 uint32_t cipStopAllInstructions ()

Stop All Instructions.

Return values:

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

ECMD_SUCCESS if successful

nonzero if unsuccessful

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

5.1.2.6 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

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

Set a hardware breakpoint in Processor using a real address.

Parameters:

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

i_address Address to set breakpoint at

i_type Type of breakpoint to set

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

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

ECMD_SUCCESS if successful

nonzero if unsuccessful

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

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

Clear a hardware breakpoint from Processor using a real address.

Parameters:

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

i_address Address to clear breakpoint at

i_type Type of breakpoint to set

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

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

ECMD_SUCCESS if successful

nonzero if unsuccessful

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

5.1.2.9 uint32_t cipGetVpr (ecmdChipTarget & *i_target*, uint32_t *i_vprNum*, ecmdDataBuffer & *o_data*)

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

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a processor

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_INVALID_ARGS Vpr 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_vprNum Number of vpr to read from

o_data DataBuffer object that holds data read from vpr

5.1.2.10 uint32_t cipGetVprMultiple (ecmdChipTarget & *i_target*, std::list< ecmdIndexEntry > & *io_entries*)

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

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a processor

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_INVALID_ARGS Vpr number is invalid

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

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

ECMD_SUCCESS if successful read

nonzero if unsuccessful

Parameters:

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

io_entries List of entries to fetch **ecmdIndexEntry.index**(p. 66) field must be filled in

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

5.1.2.11 uint32_t cipPutVpr (ecmdChipTarget & *i_target*, uint32_t *i_vprNum*, ecmdDataBuffer & *i_data*)

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

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a processor

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_INVALID_ARGS Vpr number is invalid

ECMD_SUCCESS if successful

ECMD_DATA_OVERFLOW Too much data was provided for a write

ECMD_DATA_UNDERFLOW Too little data was provided to a write function

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

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

nonzero if unsuccessful

Parameters:

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

i_vprNum Number of vpr to write to

i_data DataBuffer object that holds data to write into vpr

5.1.2.12 uint32_t cipPutVprMultiple (ecmdChipTarget & *i_target*, std::list< ecmdIndexEntry > & *i_entries*)

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

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a processor

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_INVALID_ARGS Vpr 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. 66) fields must be filled in

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

5.1.2.13 uint32_t cipGetMemProc (ecmdChipTarget & *i_target*, uint64_t *i_address*, uint32_t *i_bytes*, ecmdDataBuffer & *o_memoryData*, ecmdDataBuffer & *o_memoryTags*)

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

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a processor

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

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

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

ECMD_SUCCESS if successful read

nonzero if unsuccessful

Parameters:

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

i_address Starting address to read from

i_bytes Number of bytes to write

o_memoryData DataBuffer object that holds data read from memory

o_memoryTags 1 bit of tag for every 64 bits of memory data

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

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

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

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a processor

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

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

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

nonzero if unsuccessful

Parameters:

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

i_address Starting address to write to

i_bytes Number of bytes to write

i_memoryData DataBuffer object that holds data to write into memory

i_memoryTags 1 bit of tag for every 64 bits of memory data

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

5.1.2.15 uint32_t cipGetMemMemCtrl (ecmdChipTarget & *i_target*, uint64_t *i_address*, uint32_t *i_bytes*, ecmdDataBuffer & *o_memoryData*, ecmdDataBuffer & *o_memoryTags*)

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

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a memory controller

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

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

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

ECMD_SUCCESS if successful read

nonzero if unsuccessful

Parameters:

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

i_address Starting address to read from

i_bytes Number of bytes to write

o_memoryData DataBuffer object that holds data read from memory

o_memoryTags 1 bit of tag for every 64 bits of memory data

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

5.1.2.16 uint32_t cipPutMemMemCtrl (ecmdChipTarget & *i_target*, uint64_t *i_address*, uint32_t *i_bytes*, ecmdDataBuffer & *i_memoryData*, ecmdDataBuffer & *i_memoryTags*)

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

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a memory controller

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

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

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

nonzero if unsuccessful

Parameters:

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

i_address Starting address to write to

i_bytes Number of bytes to write

i_memoryData DataBuffer object that holds data to write into memory

i_memoryTags 1 bit of tag for every 64 bits of memory data

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

5.2 cipStructs.H File Reference

Cronus & IP eCMD Extension Structures.

Defines

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

Enumerations

- enum **ecmdBreakpointType_t** { **ECMD_BREAKPOINT_IABR**, **ECMD_BREAKPOINT_DABR**, **ECMD_BREAKPOINT_CIABR** }
  Used by setBreakpoint to specify what type of breakpoint to set.

5.2.1 Detailed Description

Cronus & IP eCMD Extension Structures.

Extension Owner : Chris Engel

5.2.2 Define Documentation

5.2.2.1 #define ECMD_CIP_CAPI_VERSION "1.1d"

eCMD CIP Extension version

5.2.3 Enumeration Type Documentation

5.2.3.1 enum ecmdBreakpointType_t

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

Enumeration values:

ECMD_BREAKPOINT_IABR Instruction Address Breakpoint.

ECMD_BREAKPOINT_DABR Data Address Breakpoint.

ECMD_BREAKPOINT_CIABR ?? Breakpoint

5.3 croClientCapi.H File Reference

Cronus eCMD Extension.

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

Load/Unload Functions

- uint32_t **croInitExtension** ()
  Initialize eCMD Cronus Extension DLL.

Misc Functions

- uint32_t **croReset** ()
  Reset Cronus internal state variables.
- uint32_t **croDisplayVersion** ()
  Display the Cronus version information to stdout.
- uint32_t **croSetDebug** (char i_major, char i_minor= '1')
  Set a Cronus debug flag -debug.
- uint32_t **croClearDebug** (char i_major, char i_minor= '1')
  Clear a Cronus debug flag -debug.
- bool **croIsDebugOn** (char i_major, char i_minor= '1')
  Checks whether Cronus debug flag is set.

5.3.1 Detailed Description

Cronus eCMD Extension.

Extension Owner : Chris Engel

5.3.2 Function Documentation

5.3.2.1 uint32_t croInitExtension ()

Initialize eCMD Cronus Extension DLL.

Return values:
ECMD_SUCCESS if successful load
ECMD_INVALID_DLL_VERSION if Dll version loaded doesn't match client version

nonzero if unsuccessful

Postcondition:
eCMD Cronus Extension is initialized and version checked

5.3.2.2 uint32_t croReset ()

Reset Cronus internal state variables.

Return values:
ECMD_SUCCESS if successful
nonzero if unsuccessful

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

5.3.2.3 uint32_t croDisplayVersion ()

Display the Cronus version information to stdout.

Return values:
ECMD_SUCCESS if successful
nonzero if unsuccessful

NOTE : This is equivalent to 'croquery version'

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

Set a Cronus debug flag -debug.

Parameters:
i_major Major debug char
i_minor Minor debug char

Return values:
ECMD_SUCCESS if successful
nonzero if unsuccessful

NOTE : To set all minor's of a particular major, set i_minor = '0' NOTE : debug 5 == 5.1 (hence the default minor number if not specified is 1

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

Clear a Cronus debug flag -debug.

Parameters:
i_major Major debug char

i_minor Minor debug char

Return values:

ECMD_SUCCESS if successful

nonzero if unsuccessful

NOTE : To clear all minor's of a particular major, set i_minor = '0' NOTE : debug 5 == 5.1 (hence the default minor number if not specified is 1

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

Checks whether Cronus debug flag is set.

Parameters:

i_major Major debug char

i_minor Minor debug char

Return values:

true if debug is on

false if debug is off

NOTE : To check if any minor of a particular major is on, set i_minor = '0' NOTE : debug 5 == 5.1 (hence the default minor number if not specified is 1

5.4 croStructs.H File Reference

Cronus eCMD Extension Structures.

Defines

- #define **ECMD_CRO_CAPI_VERSION** "1.1d"
  eCMD Cronus Extension version

5.4.1 Detailed Description

Cronus eCMD Extension Structures.

Extension Owner : Chris Engel

5.4.2 Define Documentation

5.4.2.1 #define ECMD_CRO_CAPI_VERSION "1.1d"

eCMD Cronus Extension version

5.5 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, **ecmdConfigLoopType_t** i_looptype=ECMD_SELECTED_TARGETS_LOOP)

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

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

  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 ring from the selected chip into the data buffer.

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

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

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

  Reads the selected spy into the data buffer.

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

  Writes the data buffer into the all latches matching i_latchName.

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

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

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

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

Scom Functions

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

  Scoms bits from the selected address into the data buffer.

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

  Scoms bits from the data buffer into the selected address.

Jtag Functions

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

Send a JTAG instruction and modifier to the specified chip.

FSI Functions

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

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

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

  Write data into the selected CFAM register address.

Spy Functions

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

  Reads the selected spy into the data buffer.

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

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

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

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

- uint32_t **getSpyGroups** (**ecmdChipTarget** &i_target, const char *i_spyName, std::list< **ecmdSpyGroupData** > &o_groups)

  Reads the selected spy and load all the spy groups into provided list.

- uint32_t **putSpy** (**ecmdChipTarget** &i_target, const char *i_spyName, **ecmdDataBuffer** &i_data)

  Writes the data buffer into the selected spy.

- uint32_t **putSpyEnum** (**ecmdChipTarget** &i_target, const char *i_spyName, const std::string i_enumValue)

  Writes the enum into the selected spy.

Ring Cache Functions

- void **ecmdEnableRingCache** ()

  Enables internal caching of read/writes of scan rings to the chip for functions like getring/getspy/getspr.

- uint32_t **ecmdDisableRingCache** ()

Disable internal caching of reads/writes of scan rings.

- uint32_t **ecmdFlushRingCache** ()

  Flush all modified data from the internal cache to the hardware, then remove all rings from cache.

- bool **ecmdIsRingCacheEnabled** ()

  Returns true/false to signify if caching is currently enabled.

Array Functions

- uint32_t **getArray** (**ecmdChipTarget** &i_target, const char *i_arrayName, **ecmdDataBuffer** &i_address, **ecmdDataBuffer** &o_data)

  Reads bits from the selected array into the data buffer.

- uint32_t **getArrayMultiple** (**ecmdChipTarget** &i_target, const char *i_arrayName, std::list< **ecmdArrayEntry** > &io_entries)

  Reads bits from multiple array addresses/elements into the list of data buffers.

- uint32_t **putArray** (**ecmdChipTarget** &i_target, const char *i_arrayName, **ecmdDataBuffer** &i_address, **ecmdDataBuffer** &i_data)

  Writes bits from the data buffer into the selected array.

- uint32_t **putArrayMultiple** (**ecmdChipTarget** &i_target, const char *i_arrayName, std::list< **ecmdArrayEntry** > &i_entries)

  Writes bits from the list of entries into the selected array.

Clock Functions

- uint32_t **ecmdQueryClockState** (**ecmdChipTarget** &i_target, const char *i_clockDomain, **ecmdClockState_t** &o_clockState)

  Query the state of the clocks for a domain.

- uint32_t **startClocks** (**ecmdChipTarget** &i_target, const char *i_clockDomain, bool i_forceState=false)

  Start the clocks in the domain specified.

- uint32_t **stopClocks** (**ecmdChipTarget** &i_target, const char *i_clockDomain, bool i_forceState=false)

  Stop the clocks in the domain specified.

iSteps Functions

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

  Run iSteps by number.

- uint32_t **iStepsByName** (std::string i_stepName)

Run a single iStep by name.

- uint32_t **iStepsByNameMultiple** (std::list< std::string > i_stepNames)

  Run multiple iSteps by name.

- uint32_t **iStepsByNameRange** (std::string i_stepNameBegin, std::string i_stepNameEnd)

  Run all iSteps by name starting with i_stepNameBegin and ending with i_stepNameEnd.

Processor Functions

- uint32_t **ecmdQueryProcRegisterInfo** (**ecmdChipTarget** &i_target, const char *i_name, **ecmdProcRegisterInfo** &o_data)

  Query Information about a Processor Register (SPR/GPR/FPR).

- uint32_t **getSpr** (**ecmdChipTarget** &i_target, const char *i_sprName, **ecmdDataBuffer** &o_data)

  Reads the selected Processor Architected Special Purpose Register (SPR) into the data buffer.

- uint32_t **getSprMultiple** (**ecmdChipTarget** &i_target, std::list< **ecmdNameEntry** > &io_entries)

  Reads the selected Processor Architected Special Purpose Register (SPR) into the data buffer.

- uint32_t **putSpr** (**ecmdChipTarget** &i_target, const char *i_sprName, **ecmdDataBuffer** &i_data)

  Writes the data buffer into the selected Processor Architected Special Purpose Register (SPR).

- uint32_t **putSprMultiple** (**ecmdChipTarget** &i_target, std::list< **ecmdNameEntry** > &i_entries)

  Writes the data buffer into the selected Processor Architected Special Purpose Register (SPR).

- uint32_t **getGpr** (**ecmdChipTarget** &i_target, uint32_t i_gprNum, **ecmdDataBuffer** &o_data)

  Reads the selected Processor Architected General Purpose Register (GPR) into the data buffer.

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

  Reads the selected Processor Architected General Purpose Register (GPR) into the data buffer.

- uint32_t **putGpr** (**ecmdChipTarget** &i_target, uint32_t i_gprNum, **ecmdDataBuffer** &i_data)

  Writes the data buffer into the selected Processor Architected General Purpose Register (GPR).

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

  Writes the data buffer into the selected Processor Architected General Purpose Register (GPR).

- uint32_t **getFpr** (**ecmdChipTarget** &i_target, uint32_t i_fprNum, **ecmdDataBuffer** &o_data)

Reads the selected Processor Architected Floating Point Register (FPR) into the data buffer.

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

  Reads the selected Processor Architected Floating Point Register (FPR) into the data buffer.

- uint32_t **putFpr** (**ecmdChipTarget** &i_target, uint32_t i_fprNum, **ecmdDataBuffer** &i_data)

  Writes the data buffer into the selected Processor Architected Floating Point Register (FPR).

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

  Writes the data buffer into the selected Processor Architected Floating Point Register (FPR).

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 using a real address.

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

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

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

  Reads System Mainstore through the PSI or DMA interface (whichever is avialable) using a real address.

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

  Writes System Mainstore through the PSI or DMA interface (whichever is avialable) using a real address.

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

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

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

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

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 value.

- uint32_t **simpolltcfac** (const char *i_tcfacname, **ecmdDataBuffer** &i_expect, uint32_t i_row=0, uint32_t i_startbit=0, uint32_t i_bitlength=0, uint32_t i_maxcycles=1, uint32_t i_pollinterval=1)

  Poll a TCFAC facility waiting for expected value.

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

  Write a Facility using a name.

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

  Write a Facility using a name - preserving Xstate.

- uint32_t **simputtcfac** (const char *i_tcfacname, uint32_t i_bitlength, **ecmdDataBuffer** &i_data, uint32_t i_row=0, uint32_t i_numrows=0)

  Write a TCFAC facility.

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

  Load a checkpoint into model.

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

  Stick a Facility using a name.

- uint32_t **simstktcfac** (const char *i_tcfacname, uint32_t i_bitlength, **ecmdDataBuffer** &i_data, uint32_t i_row=0, uint32_t i_numrows=0)

  Stick a TCFAC facility.

- uint32_t **simSUBCMD** (const char *i_command)

  Run RTX SUBCMD.

- uint32_t **simtckinterval** (uint32_t i_tckinterval)

  Set TCK Interval setting in the model for JTAG Master.

- uint32_t **simUNSTICK** (const char *i_facname, uint32_t i_bitlength, uint32_t i_row=0, uint32_t i_offset=0)

  Unstick a Facility using a name.

- uint32_t **simunsticktcfac** (const char *i_tcfacname, uint32_t i_bitlength, **ecmdDataBuffer** &i_data, uint32_t i_row=0, uint32_t i_numrows=0)

  Unstick a TCFAC facility.

- uint32_t **simGetHierarchy** (**ecmdChipTarget** &i_target, std::string &o_hierarchy)

  Fetch the hierarchy for the specified chip target relative to the latch names in the scandef.

- uint32_t **ecmdQueryChipSimModelVersion** (**ecmdChipTarget** &i_target, std::string &o_timestamp)

  Will retrieve the model timestamp from the simulation, in hardware mode "NA" is returned.

- uint32_t **ecmdQueryChipScandefVersion** (**ecmdChipTarget** &i_target, std::string &o_timestamp)

  Will retrieve the scandef timestamp from the scandef being used for the specified target.

Error Handling Functions

- std::string **ecmdGetErrorMsg** (uint32_t i_errorCode, bool i_parseReturnCode=true)

  Retrieve additional error information for errorcode.

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

  Register an Error Message that has occured.

Output Functions

- void **ecmdOutputError** (const char *i_message)

  Output a message related to an error.

- void **ecmdOutputWarning** (const char *i_message)

  Output a message related to an warning.

- void **ecmdOutput** (const char *i_message)

  Output a message to the screen or logs.

Misc Functions

- uint32_t **ecmdGetGlobalVar** (**ecmdGlobalVarType_t** i_type)

  Retrieve the value of some ecmdGlobalVars.

- void **ecmdSetTraceMode** (**ecmdTraceType_t** i_type, bool i_enable)

  Enable/Disable a trace mode.

- bool **ecmdQueryTraceMode** (**ecmdTraceType_t** i_type)

  Query the state of a trace mode.

- uint32_t **ecmdDelay** (uint32_t i_simCycles, uint32_t i_msDelay)

  Function to delay a procedure either by running sim cycles or by doing a millisecond delay.

- uint32_t **makeSPSystemCall** (**ecmdChipTarget** &i_target, const std::string &i_command, std::string &o_stdout)

  Make a system call on the targetted Service Processor or Service Element.

Configuration Functions

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

  Retrieve the value of a Configuration Setting.

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

  Set the value of a Configuration Setting.

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

  Deconfigure a target in the system.

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

  Configure a target in the system - must be previously known to the system.

- uint32_t **ecmdTargetToUnitId** (**ecmdChipTarget** &io_target)

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

- uint32_t **ecmdUnitIdStringToTarget** (std::string i_unitId, std::list< **ecmdChipTarget** > &o_targetList)

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

- uint32_t **ecmdUnitIdToTarget** (uint32_t i_unitId, std::list< **ecmdChipTarget** > &o_targetList)

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

5.5.1 Detailed Description

eCMD C/C++ Client Interface

5.5.2 Function Documentation

5.5.2.1 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.

5.5.2.2 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

5.5.2.3 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

5.5.2.4 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

5.5.2.5 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

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

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

Parameters:

io_target Struct that contains partial information to limit query results - chipType is unused

o_queryData Return data from query

i_looptype (Optional) Used by config looper to specify different query modes

Return values:

ECMD_SUCCESS if successful

nonzero on failure

This function acts just like ecmdQueryConfig except it operates on what targets were selected by the user args -n#, -p#, -c#, -t#

Use of this function is the same as ecmdQueryConfig

When -talive is specified all threads configured will be returned in o_queryData and io_target.threadState will be set to ECMD_TARGET_THREAD_ALIVE. NOTE : This function does not affect ring caching

5.5.2.7 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

5.5.2.8 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

5.5.2.9 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

5.5.2.10 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

5.5.2.11 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

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

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

Return values:

ECMD_INVALID_RING if ringname is not valid for target

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

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

ECMD_SUCCESS if successful

nonzero if unsuccessful

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

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core information of ring to read

i_ringName Name of ring to read from

o_data DataBuffer object that holds data read from ring

See also:

putRing(p. 118)

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

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

Return values:

ECMD_SUCCESS if successful

nonzero if unsuccessful

ECMD_DATA_OVERFLOW Too much data was provided for a write

ECMD_DATA_UNDERFLOW Too little data was provided to a write function

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_INVALID_RING if ringname is not valid for target

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

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

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core information of ring to write

i_ringName Name of ring to write to

i_data DataBuffer object that holds data to write into ring

See also:

getRing(p. 117)

5.5.2.14 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

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

Writes the data buffer into the all latches matching i_latchName.

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

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

ECMD_UNABLE_TO_OPEN_SCANDEF eCMD was unable to open the scandef

ECMD_INVALID_RING if ringname is not valid for target

ECMD_INVALID_LATCHNAME if latchname not found in scandef

ECMD_DATA_OVERFLOW Too much data was provided for a write

ECMD_DATA_UNDERFLOW Too little data was provided to a write function

nonzero if unsuccessful

Parameters:

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

i_latchName Name of latch to write (can be a partial or full name based on i_mode)

i_data DataBuffer object that holds data to write into latch

i_ringName Name of ring to search for latch if == NULL, entire scandef is searched

i_mode LatchName search mode

i_startBit Startbit in latchname to insert data

i_numBits Number of bits to insert from startbit

o_matchs Number of latchs found that matched your name and data was inserted

NOTE : This function is ring cache enabled

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

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

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

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

ECMD_SUCCESS if successful

nonzero if unsuccessful

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

Parameters:

i_target Struct that contains chip and cage/node/slot/position information of ring to read

i_ address Address of ring to read from

i_ bitLength Bit Length to scan for

o_ data DataBuffer object that holds data read from ring

See also:
putRingWithModifier(p. 120)

NOTE : This is a debug interface and should not be used in normal situations NOTE : This function does not handle processor cores for you, the i_address will be taken and used with no modifications so you are responsible for specifying the correct core address NOTE : This function will only scan for the length provided, if this length doesn't match the actual length of the ring corruption may occur

5.5.2.17 uint32_t putRingWithModifier (ecmdChipTarget & *i_ target*, uint32_t *i_ address*, uint32_t *i_ bitLength*, ecmdDataBuffer & *i_ data*)

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

Return values:
ECMD_ SUCCESS if successful

nonzero if unsuccessful

ECMD_ DATA_ OVERFLOW Too much data was provided for a write

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

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

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

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

Parameters:
i_ target Struct that contains chip and cage/node/slot/position information of ring to write

i_ address Address of ring to write to

i_ bitLength Bit Length to scan for

i_ data DataBuffer object that holds data to write into ring

See also:
getRingWithModifier(p. 119)

NOTE : This is a debug interface and should not be used in normal situations NOTE : This function does not handle processor cores for you, the i_address will be taken and used with no modifications so you are responsible for specifying the correct core address NOTE : This function will only scan for the length provided, if this length doesn't match the actual length of the ring corruption may occur

5.5.2.18 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. 121)

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

5.5.2.19 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. 121)

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

5.5.2.20 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

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

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

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

nonzero if unsuccessful

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

ECMD_NON_FSI_CHIP Targetted chip is not attached via FSI

Parameters:

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

i_address CFAM address to read from

o_data DataBuffer object that holds data read from address

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

Write data into the selected CFAM register address.

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_DATA_OVERFLOW Too much data was provided for a write

ECMD_DATA_UNDERFLOW Too little data was provided to a write function

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

ECMD_SUCCESS if successful

ECMD_NON_FSI_CHIP Targetted chip is not attached via FSI

nonzero if unsuccessful

Parameters:

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

i_address CFAM address to write to

i_data DataBuffer object that holds data to write into address

5.5.2.23 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

5.5.2.24 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

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

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

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

ECMD_INVALID_SPY Spy name is invalid or Spy is not an ECC Grouping

ECMD_SPY_FAILED_ECC_CHECK if invalid ECC detected on Spy read - valid Spy Data still returned

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

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core information of spy to read

i_spyEpCheckersName Name of spy to read from

o_inLatchData Return the data for the input to the eccGroup

o_outLatchData Return the Ecc data associated with the outbits of the eccGroup

o_eccErrorMask Return a mask for the Ecc data a 1 in the mask means the associated eccData was in error

Return values:

nonzero if unsuccessful

NOTE : This function is ring cache enabled

5.5.2.26 uint32_t getSpyGroups (ecmdChipTarget & *i_target*, const char * *i_spyName*, std::list< ecmdSpyGroupData > & *o_groups*)

Reads the selected spy and load all the spy groups into provided list.

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SPY_FAILED_ECC_CHECK if invalid ECC detected on Spy read

ECMD_INVALID_SPY Spy name is invalid or Spy is an ECC Grouping

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

ECMD_SPY_IS_EDIAL Spy is an edial have to use getSpyEnum

ECMD_SUCCESS if successful read

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core information of spy to read

i_spyName Name of spy to read from

o_groups List of structures containing the group data and deadbits mask

NOTE : This function is ring cache enabled

5.5.2.27 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

5.5.2.28 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

5.5.2.29 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

5.5.2.30 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

5.5.2.31 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

5.5.2.32 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

5.5.2.33 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. 128)
getArrayMultiple(p. 128)

5.5.2.34 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. 128)
getArray(p. 127)

NOTE : To use this function the io_entries list should be pre-loaded with the addresses to fetch, the associated dataBuffers will be loaded upon return

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

5.5.2.35 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. 127)

5.5.2.36 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. 127)

NOTE : i_entries should be pre-loaded with address and data

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

5.5.2.37 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

5.5.2.38 uint32_t startClocks (ecmdChipTarget & *i_target*, const char * *i_clockDomain*, bool *i_forceState* = false)

Start the clocks in the domain specified.

Return values:

ECMD_SUCCESS if successful

nonzero if unsuccessful

ECMD_INVALID_CLOCK_DOMAIN An invalid clock domain name was specified

ECMD_CLOCKS_ALREADY_ON The clocks in the specified domain are already on

ECMD_CLOCKS_IN_INVALID_STATE The clock in the specified domain are in an unknown state (not all on/off)

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

Parameters:

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

i_clockDomain Clock domain to start - as defined in scandef - use "ALL" to start all domains

i_forceState Force the clocks into the appropriate state - ignore if not in correct state to start

NOTE : If i_target refers to a particular chip object the i_clockDomain has to be "ALL" or a clock domain as defined in the scandef If i_target refers to a Cage/node then i_clockDomain has to be "ALL" or one of the predefined convenience clock domains as documented in the eCMD system spec for your particular product.

5.5.2.39 uint32_t stopClocks (ecmdChipTarget & *i_target*, const char * *i_clockDomain*, bool *i_forceState* = false)

Stop the clocks in the domain specified.

Return values:

ECMD_SUCCESS if successful

nonzero if unsuccessful

ECMD_INVALID_CLOCK_DOMAIN An invalid clock domain name was specified

ECMD_CLOCKS_ALREADY_OFF The clocks in the specified domain are already off

ECMD_CLOCKS_IN_INVALID_STATE The clock in the specified domain are in an unknown state (not all on/off)

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

Parameters:

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

i_clockDomain Clock domain to stop - as defined in scandef - use "ALL" to stop all domains

i_forceState Force the clocks into the appropriate state - ignore if not in correct state to start

NOTE : If i_target refers to a particular chip object the i_clockDomain has to be "ALL" or a clock domain as defined in the scandef If i_target refers to a Cage/node then i_clockDomain has to be "ALL" or one of the predefined convenience clock domains as documented in the eCMD system spec for your particular product.

5.5.2.40 uint32_t iStepsByNumber (ecmdDataBuffer & *i_steps*)

Run iSteps by number.

Return values:

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

ECMD_ISTEPS_INVALID_STEP An invalid step number was provided

ECMD_SUCCESS if successful

nonzero if unsuccessful

Postcondition:

iSteps specified are complete

Parameters:

i_steps Bit mask defining which steps to run

NOTE - function returns on first failure and remaining steps are not run

5.5.2.41 uint32_t iStepsByName (std::string *i_stepName*)

Run a single iStep by name.

Return values:

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

ECMD_ISTEPS_INVALID_STEP An invalid step name was provided

ECMD_SUCCESS if successful

nonzero if unsuccessful

Postcondition:
iStep specified is complete

Parameters:
i_stepName List of iStep names to run

5.5.2.42 uint32_t iStepsByNameMultiple (std::list< std::string > *i_stepNames*)

Run multiple iSteps by name.

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

ECMD_ISTEPS_INVALID_STEP An invalid step name was provided

ECMD_SUCCESS if successful

nonzero if unsuccessful

Postcondition:
iSteps specified are complete

Parameters:
i_stepNames List of iStep names to run

NOTE - Steps are run in order as is appropriate for proper system configuration, not by order provided in list NOTE - function returns on first failure and remaining steps are not run

5.5.2.43 uint32_t iStepsByNameRange (std::string *i_stepNameBegin*, std::string *i_stepNameEnd*)

Run all iSteps by name starting with i_stepNameBegin and ending with i_stepNameEnd.

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

ECMD_ISTEPS_INVALID_STEP An invalid step name was provided

ECMD_SUCCESS if successful

nonzero if unsuccessful

Postcondition:
iSteps specified are complete

Parameters:
i_stepNameBegin Starting iStep to run

i_stepNameEnd Ending iStep to run

NOTE - function returns on first failure and remaining steps are not run

5.5.2.44 uint32_t ecmdQueryProcRegisterInfo (ecmdChipTarget & *i_target*, const char * *i_name*, ecmdProcRegisterInfo & *o_data*)

Query Information about a Processor Register (SPR/GPR/FPR).

Parameters:

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

i_name Name of the Register to fetch data about (can be either a specific SPR or GPR/FPR)

o_data Data retrieved about the register

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a processor

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_INVALID_SPR Spr name is invalid

ECMD_SUCCESS if successful read

nonzero if unsuccessful

5.5.2.45 uint32_t getSpr (ecmdChipTarget & *i_target*, const char * *i_sprName*, ecmdDataBuffer & *o_data*)

Reads the selected Processor Architected Special Purpose Register (SPR) into the data buffer.

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a processor

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_INVALID_SPR Spr name is invalid

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

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

ECMD_SUCCESS if successful read

nonzero if unsuccessful

Parameters:

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

i_sprName Name of spr to read from

o_data DataBuffer object that holds data read from spr

5.5.2.46 uint32_t getSprMultiple (ecmdChipTarget & *i_target*, std::list< ecmdNameEntry > & *io_entries*)

Reads the selected Processor Architected Special Purpose Register (SPR) into the data buffer.

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a processor

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_INVALID_SPR Spr name is invalid

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

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

ECMD_SUCCESS if successful read

nonzero if unsuccessful

Parameters:

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

io_entries List of entries to fetch **ecmdNameEntry.name**(p. 72) field must be filled in

- NOTE : There are special keywords that can be specified to fetch groups of entries, they are used by adding only an entry to io_entries and setting **ecmdNameEntry.name**(p. 72) =
  - "ALLTHREADED" : To fetch all threaded (replicated) SPR's for particular target
  - "ALLSHARED" : To fetch all non-threaded SPR's for particular target

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

5.5.2.47 uint32_t putSpr (ecmdChipTarget & *i_target*, const char * *i_sprName*, ecmdDataBuffer & *i_data*)

Writes the data buffer into the selected Processor Architected Special Purpose Register (SPR).

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a processor

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

ECMD_INVALID_SPR Spr name is invalid

ECMD_DATA_OVERFLOW Too much data was provided for a write

ECMD_DATA_UNDERFLOW Too little data was provided to a write function

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

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

nonzero if unsuccessful

Parameters:

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

i_sprName Name of spr to write to

i_data DataBuffer object that holds data to write into spr

5.5.2.48 uint32_t putSprMultiple (ecmdChipTarget & *i_target*, std::list< ecmdNameEntry > & *i_entries*)

Writes the data buffer into the selected Processor Architected Special Purpose Register (SPR).

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a processor

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

ECMD_INVALID_SPR Spr name is invalid

ECMD_DATA_OVERFLOW Too much data was provided for a write

ECMD_DATA_UNDERFLOW Too little data was provided to a write function

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

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

nonzero if unsuccessful

Parameters:

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

i_entries List of entries to write all **ecmdNameEntry**(p. 72) fields must be filled in

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

5.5.2.49 uint32_t getGpr (ecmdChipTarget & *i_target*, uint32_t *i_gprNum*, ecmdDataBuffer & *o_data*)

Reads the selected Processor Architected General Purpose Register (GPR) into the data buffer.

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a processor

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_INVALID_GPR Gpr number is invalid

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

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

ECMD_SUCCESS if successful read

nonzero if unsuccessful

Parameters:

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

i_gprNum Number of gpr to read from

o_data DataBuffer object that holds data read from gpr

5.5.2.50 uint32_t getGprMultiple (ecmdChipTarget & *i_target*, std::list< ecmdIndexEntry > & *io_entries*)

Reads the selected Processor Architected General Purpose Register (GPR) into the data buffer.

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a processor

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_INVALID_GPR Gpr number is invalid

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

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

ECMD_SUCCESS if successful read

nonzero if unsuccessful

Parameters:

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

io_entries List of entries to fetch **ecmdIndexEntry.index**(p. 66) field must be filled in

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

5.5.2.51 uint32_t putGpr (ecmdChipTarget & *i_target*, uint32_t *i_gprNum*, ecmdDataBuffer & *i_data*)

Writes the data buffer into the selected Processor Architected General Purpose Register (GPR).

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a processor

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_INVALID_GPR Gpr number is invalid

ECMD_SUCCESS if successful

ECMD_DATA_OVERFLOW Too much data was provided for a write

ECMD_DATA_UNDERFLOW Too little data was provided to a write function

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

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

nonzero if unsuccessful

Parameters:

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

i_gprNum Number of gpr to write to

i_data DataBuffer object that holds data to write into gpr

5.5.2.52 uint32_t putGprMultiple (ecmdChipTarget & *i_target*, std::list< ecmdIndexEntry > & *i_entries*)

Writes the data buffer into the selected Processor Architected General Purpose Register (GPR).

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a processor

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_INVALID_GPR Gpr number is invalid

ECMD_SUCCESS if successful

ECMD_DATA_OVERFLOW Too much data was provided for a write

ECMD_DATA_UNDERFLOW Too little data was provided to a write function

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

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

nonzero if unsuccessful

Parameters:

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

i_entries List of entries to write all **ecmdIndexEntry**(p. 66) fields must be filled in

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

5.5.2.53 uint32_t getFpr (ecmdChipTarget & *i_target*, uint32_t *i_fprNum*, ecmdDataBuffer & *o_data*)

Reads the selected Processor Architected Floating Point Register (FPR) into the data buffer.

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a processor

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

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

ECMD_INVALID_FPR Fpr number is invalid

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

ECMD_SUCCESS if successful read

nonzero if unsuccessful

Parameters:

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

i_fprNum Number of fpr to read from

o_data DataBuffer object that holds data read from fpr

5.5.2.54 uint32_t getFprMultiple (ecmdChipTarget & *i_target*, std::list< ecmdIndexEntry > & *io_entries*)

Reads the selected Processor Architected Floating Point Register (FPR) into the data buffer.

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a processor

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_INVALID_FPR Fpr number is invalid

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

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

ECMD_SUCCESS if successful read

nonzero if unsuccessful

Parameters:

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

io_entries List of entries to fetch **ecmdIndexEntry.index**(p. 66) field must be filled in

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

5.5.2.55 uint32_t putFpr (ecmdChipTarget & *i_target*, uint32_t *i_fprNum*, ecmdDataBuffer & *i_data*)

Writes the data buffer into the selected Processor Architected Floating Point Register (FPR).

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a processor

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

ECMD_INVALID_FPR Fpr number is invalid

ECMD_DATA_OVERFLOW Too much data was provided for a write

ECMD_DATA_UNDERFLOW Too little data was provided to a write function

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

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

nonzero if unsuccessful

Parameters:

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

i_fprNum Number of fpr to write to

i_data DataBuffer object that holds data to write into fpr

5.5.2.56 uint32_t putFprMultiple (ecmdChipTarget & *i_target*, std::list< ecmdIndexEntry > & *i_entries*)

Writes the data buffer into the selected Processor Architected Floating Point Register (FPR).

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a processor

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_INVALID_FPR Fpr number is invalid

ECMD_SUCCESS if successful

ECMD_DATA_OVERFLOW Too much data was provided for a write

ECMD_DATA_UNDERFLOW Too little data was provided to a write function

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

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

nonzero if unsuccessful

Parameters:

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

i_entries List of entries to write all **ecmdIndexEntry**(p. 66) fields must be filled in

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

5.5.2.57 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

5.5.2.58 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 entry that caused the failure in the list will also be marked with the same return code. That data and all subsequent entries in the list will not be fetched and the data should be considered invalid.

- NOTE : to fetch all Trace Arrays available add only one entry to io_entries and set **ecmdNameVectorEntry.name**(p. 73) = "ALL"

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

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

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a processor

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

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

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

ECMD_SUCCESS if successful read

nonzero if unsuccessful

Parameters:

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

i_address Starting address to read from

i_bytes Number of bytes to write

o_data DataBuffer object that holds data read from memory

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

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

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a processor

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

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

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

nonzero if unsuccessful

Parameters:

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

i_ address Starting address to write to

i_ bytes Number of bytes to write

i_ data DataBuffer object that holds data to write into memory

5.5.2.61 uint32_t getMemDma (ecmdChipTarget & *i_ target*, uint64_t *i_ address*, uint32_t *i_ bytes*, ecmdDataBuffer & *o_ data*)

Reads System Mainstore through the PSI or DMA interface (whichever is avialable) using a real address.

Return values:

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

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

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

ECMD_ SUCCESS if successful read

nonzero if unsuccessful

Parameters:

i_ target Struct that contains cage/node information

i_ address Starting address to read from

i_ bytes Number of bytes to write

o_ data DataBuffer object that holds data read from memory

5.5.2.62 uint32_t putMemDma (ecmdChipTarget & *i_ target*, uint64_t *i_ address*, uint32_t *i_ bytes*, ecmdDataBuffer & *i_ data*)

Writes System Mainstore through the PSI or DMA interface (whichever is avialable) using a real address.

Return values:

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

ECMD_ SUCCESS if successful

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

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

nonzero if unsuccessful

Parameters:

i_ target Struct that contains cage/node information

i_ address Starting address to write to

i_ bytes Number of bytes to write

i_ data DataBuffer object that holds data to write into memory

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

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

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a memory controller

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

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

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

ECMD_SUCCESS if successful read

nonzero if unsuccessful

Parameters:

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

i_address Starting address to read from

i_bytes Number of bytes to write

o_data DataBuffer object that holds data read from memory

WARNING : This operation is typically not cache-coherent

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

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

Return values:

ECMD_TARGET_INVALID_TYPE if target is not a memory controller

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

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

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

nonzero if unsuccessful

Parameters:

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

i_address Starting address to write to

i_bytes Number of bytes to write

i_data DataBuffer object that holds data to write into memory

WARNING : This operation is typically not cache-coherent

5.5.2.65 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

5.5.2.66 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

5.5.2.67 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

5.5.2.68 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

5.5.2.69 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

5.5.2.70 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

5.5.2.71 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

5.5.2.72 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

5.5.2.73 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

5.5.2.74 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

5.5.2.75 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

5.5.2.76 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

5.5.2.77 uint32_t simPOLLFAC (const char * *i_facname*, uint32_t *i_bitlength*, ecmdDataBuffer & *i_expect*, uint32_t *i_row* = 0, uint32_t *i_offset* = 0, uint32_t *i_maxcycles* = 1, uint32_t *i_pollinterval* = 1)

Poll a facility waiting for expected value.

Parameters:

i_facname Facility name

i_bitlength Bit length to expect

i_expect Data to expect in facility

i_row Optional: Array row

i_offset Optional : Facility offset

i_maxcycles Optional : Maximum number of cycles to run

i_pollinterval Option : Number of clock cycles to run between each poll

Return values:

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

ECMD_POLLING_FAILURE Polling completed without reaching expected value

ECMD_SUCCESS if successful

nonzero on failure

5.5.2.78 uint32_t simpolltcfac (const char * *i_tcfacname*, ecmdDataBuffer & *i_expect*, uint32_t *i_row* = 0, uint32_t *i_startbit* = 0, uint32_t *i_bitlength* = 0, uint32_t *i_maxcycles* = 1, uint32_t *i_pollinterval* = 1)

Poll a TCFAC facility waiting for expected value.

Parameters:

i_tcfacname Facility name

i_bitlength Bit length to expect

i_expect Data to expect in facility

i_row Optional: Array row

i_startbit Optional : Facility startbit

i_maxcycles Optional : Maximum number of cycles to run

i_pollinterval Option : Number of clock cycles to run between each poll

Return values:

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

ECMD_POLLING_FAILURE Polling completed without reaching expected value

ECMD_SUCCESS if successful

nonzero on failure

5.5.2.79 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

5.5.2.80 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

5.5.2.81 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

5.5.2.82 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

5.5.2.83 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

5.5.2.84 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

5.5.2.85 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

5.5.2.86 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

5.5.2.87 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

5.5.2.88 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

5.5.2.89 uint32_t simGetHierarchy (ecmdChipTarget & *i_target*, std::string & *o_hierarchy*)

Fetch the hierarchy for the specified chip target relative to the latch names in the scandef.

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

nonzero if unsuccessful

Parameters:

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

o_hierarchy Return the model hierarchy for this target

NOTE - To retrieve the hierarchy of a processor core the core field must be set and the state set to ECMD_TARGET_QUERY_FIELD_VALID

5.5.2.90 uint32_t ecmdQueryChipSimModelVersion (ecmdChipTarget & *i_target*, std::string & *o_timestamp*)

Will retrieve the model timestamp from the simulation, in hardware mode "NA" is returned.

Parameters:

i_target Target to query for information

o_timestamp Timestamp value from simulation model

Return values:

ECMD_SUCCESS on success

non-zero on failure

5.5.2.91 uint32_t ecmdQueryChipScandefVersion (ecmdChipTarget & *i_target*, std::string & *o_timestamp*)

Will retrieve the scandef timestamp from the scandef being used for the specified target.

Parameters:

i_target Target to query for information

o_timestamp Timestamp value from scandef

Return values:

ECMD_SUCCESS on success

non-zero on failure

5.5.2.92 std::string ecmdGetErrorMsg (uint32_t *i_errorCode*, bool *i_parseReturnCode* = true)

Retrieve additional error information for errorcode.

Parameters:

i_errorCode Error code to lookup up message for

i_parseReturnCode If true will search through return codes definitions to return define name of error code

Return values:

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

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

Register an Error Message that has occured.

5.5.2.94 void ecmdOutputError (const char * *i_message*)

Output a message related to an error.

Parameters:

i_message String to output

5.5.2.95 void ecmdOutputWarning (const char * *i_message*)

Output a message related to an warning.

Parameters:

i_message String to output

5.5.2.96 void ecmdOutput (const char * *i_message*)

Output a message to the screen or logs.

Parameters:

i_message String to output

5.5.2.97 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

5.5.2.98 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

5.5.2.99 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

5.5.2.100 uint32_t ecmdDelay (uint32_t *i_simCycles*, uint32_t *i_msDelay*)

Function to delay a procedure either by running sim cycles or by doing a millisecond delay.

Parameters:

i_simCycles Number of sim cycles to run in simulation mode

i_msDelay Number of milliseconds to delay in hardware mode

Return values:

ECMD_SUCCESS on success

non-zero on failure

5.5.2.101 uint32_t makeSPSystemCall (ecmdChipTarget & *i_target*, const std::string & *i_command*, std::string & *o_stdout*)

Make a system call on the targetted Service Processor or Service Element.

Parameters:

i_target SP to run command on

i_command Command line call to make

o_stdout Standard out captured by running command

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

Retrieve the value of a Configuration Setting.

Parameters:

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

i_ name Name of setting as defined by eCMD Api

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

o_ valueAlpha Alpha value of setting (if appropriate)

o_ valueNumeric Numeric value of setting (if appropriate)

Return values:

ECMD_ INVALID_ CONFIG_ NAME Name specified is not valid

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

ECMD_ SUCCESS if successful

5.5.2.103 uint32_t ecmdSetConfiguration (ecmdChipTarget & *i_ target*, std::string *i_ name*, ecmdConfigValid_t *i_ validInput*, std::string *i_ valueAlpha*, uint32_t *i_ valueNumeric*)

Set the value of a Configuration Setting.

Parameters:

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

i_ name Name of setting as defined by eCMD Api

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

i_ valueAlpha Alpha value of setting (if appropriate)

i_ valueNumeric Numeric value of setting (if appropriate)

Return values:

ECMD_ DBUF_ INVALID_ DATA_ FORMAT Value is not in correct format for specified configuration setting

ECMD_ INVALID_ CONFIG_ NAME Name specified is not valid

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

ECMD_ SUCCESS if successful

nonzero on failure

5.5.2.104 uint32_t ecmdDeconfigureTarget (ecmdChipTarget & *i_ target*)

Deconfigure a target in the system.

Parameters:

i_ target Target info to specify what to deconfigure (target states must be set)

Return values:

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

ECMD_ SUCCESS if successful

nonzero on failure

NOTE - lowest state that is valid is level that is deconfigured. ex - if coreState is VALID the core selected is deconfigured ex - if coreState is UNUSED and posState is VALID then the pos is deconfigured

This interface allows you to deconfigure all levels cages, nodes, slots, pos's, cores

5.5.2.105 uint32_t ecmdConfigureTarget (ecmdChipTarget & *i_target*)

Configure a target in the system - must be previously known to the system.

Parameters:

i_target Target info to specify what to configure (target states must be set)

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system, or was not previously deconfigured

ECMD_SUCCESS if successful

nonzero on failure

NOTE - lowest state that is valid is level that is configured. ex - if coreState is VALID the core selected is configured ex - if coreState is UNUSED and posState is VALID then the pos is configured

This interface allows you to configure all levels cages, nodes, slots, pos's, cores

5.5.2.106 uint32_t ecmdTargetToUnitId (ecmdChipTarget & *io_target*)

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

Parameters:

io_target an **ecmdChipTarget**(p. 19) struct representing a specific eCmd target

Return values:

ECMD_SUCCESS if conversion successful

ECMD_INVALID_ARGS if unsuccessful in finding a matching Unit ID

Postcondition:

HOM Unit Ids in **ecmdChipTarget**(p. 19) struct are set and valid

5.5.2.107 uint32_t ecmdUnitIdStringToTarget (std::string *i_unitId*, std::list< ecmdChipTarget > & *o_targetList*)

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

Parameters:

i_unitId a string representing the name of a unitId

o_targetList a list of targets that match the input unitId string

Return values:

ECMD_SUCCESS if conversion successful

ECMD_INVALID_ARGS if unsuccessful in matching the string to a target

Postcondition:

There will be a list ecmdChipTargets that represent the passed in unitId string

5.5.2.108 uint32_t ecmdUnitIdToTarget (uint32_t *i_unitId*, std::list< ecmdChipTarget > & *o_targetList*)

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

Parameters:

i_unitId a uint32_t representing an unitID

o_targetList a list of targets that match the unitId input

Return values:

ECMD_SUCCESS if conversion successful

ECMD_INVALID_ARGS if unsuccessful in matching the string to a target

Postcondition:

ecmdChipTarget(p. 19) Fields are set and represent the passed in unitId string

5.6 ecmdDataBuffer.H File Reference

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

```
#include <string>
```

```
#include <inttypes.h>
```

Classes

- class **ecmdDataBufferImplementationHelper**

  This is used to help low-level implementation of the **ecmdDataBuffer**(p. 25)*, this CAN NOT be used by any eCMD client or data corruption will occur.*

- class **ecmdDataBuffer**

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

Defines

- #define **ECMD_DBUF_SUCCESS** 0x0

  DataBuffer returned successfully.

- #define **ECMD_DBUF_INIT_FAIL** (0x01000000 | 0x2000)

  Initialization of the DataBuffer failed.

- #define **ECMD_DBUF_BUFFER_OVERFLOW** (0x01000000 | 0x2010)

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

- #define **ECMD_DBUF_XSTATE_ERROR** (0x01000000 | 0x2020)

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

- #define **ECMD_DBUF_UNDEFINED_FUNCTION** (0x01000000 | 0x2030)

  Function not included in this version of DataBuffer.

- #define **ECMD_DBUF_INVALID_ARGS** (0x01000000 | 0x2040)

  Args provided to dataBuffer were invalid.

- #define **ECMD_DBUF_INVALID_DATA_FORMAT** (0x01000000 | 0x2041)

  String data didn't match expected input format.

- #define **ECMD_DBUF_FOPEN_FAIL** (0x01000000 | 0x2050)

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

- #define **ECMD_DBUF_FILE_FORMAT_MISMATCH** (0x01000000 | 0x2051)

  In readFile specified format not found in the data file.

- #define **ECMD_DBUF_NOT_OWNER** (0x01000000 | 0x2060)

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

- #define **ETRAC0**(fmt) printf("%s> ETRC: " fmt "\n", __FUNCTION__);
- #define **ETRAC1**(fmt, arg1) printf("%s> ETRC: " fmt "\n", __FUNCTION__, arg1);
- #define **ETRAC2**(fmt, arg1, arg2) printf("%s> ETRC: " fmt "\n", __FUNCTION__, arg1, arg2);
- #define **ETRAC3**(fmt, arg1, arg2, arg3) printf("%s> ETRC: " fmt "\n", __FUNCTION__, arg1, arg2, arg3);
- #define **ETRAC4**(fmt, arg1, arg2, arg3, arg4) printf("%s> ETRC: " fmt "\n", __FUNCTION__, arg1, arg2, arg3, arg4);
- #define **ETRAC5**(fmt, arg1, arg2, arg3, arg4, arg5) printf("%s> ETRC: " fmt "\n", __FUNCTION__, arg1, arg2, arg3, arg4, arg5);
- #define **ETRAC6**(fmt, arg1, arg2, arg3, arg4, arg5, arg6) printf("%s> ETRC: " fmt "\n", __FUNCTION__, arg1, arg2, arg3, arg4, arg5, arg6);
- #define **ETRAC7**(fmt, arg1, arg2, arg3, arg4, arg5, arg6, arg7) printf("%s> ETRC: " fmt "\n", __FUNCTION__, arg1, arg2, arg3, arg4, arg5, arg6, arg7);
- #define **ETRAC8**(fmt, arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8) printf("%s> ETRC: " fmt "\n", __FUNCTION__, arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8);
- #define **ETRAC9**(fmt, arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9) printf("%s> ETRC: " fmt "\n", __FUNCTION__, arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9);

Enumerations

- enum **ecmdFormatType_t** { **ECMD_SAVE_FORMAT_BINARY**, **ECMD_SAVE_FORMAT_BINARY_DATA**, **ECMD_SAVE_FORMAT_ASCII**, **ECMD_SAVE_FORMAT_XSTATE** }

  This is the different formats in which the output file will be written.

5.6.1 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

5.6.2 Define Documentation

5.6.2.1 #define ECMD_DBUF_SUCCESS 0x0

DataBuffer returned successfully.

5.6.2.2 #define ECMD_DBUF_INIT_FAIL (0x01000000 | 0x2000)

Initialization of the DataBuffer failed.

5.6.2.3 #define ECMD_DBUF_BUFFER_OVERFLOW (0x01000000 | 0x2010)

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

5.6.2.4 #define ECMD_DBUF_XSTATE_ERROR (0x01000000 | 0x2020)

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

5.6.2.5 #define ECMD_DBUF_UNDEFINED_FUNCTION (0x01000000 | 0x2030)

Function not included in this version of DataBuffer.

5.6.2.6 #define ECMD_DBUF_INVALID_ARGS (0x01000000 | 0x2040)

Args provided to dataBuffer were invalid.

5.6.2.7 #define ECMD_DBUF_INVALID_DATA_FORMAT (0x01000000 | 0x2041)

String data didn't match expected input format.

5.6.2.8 #define ECMD_DBUF_FOPEN_FAIL (0x01000000 | 0x2050)

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

5.6.2.9 #define ECMD_DBUF_FILE_FORMAT_MISMATCH (0x01000000 | 0x2051)

In readFile specified format not found in the data file.

5.6.2.10 #define ECMD_DBUF_NOT_OWNER (0x01000000 | 0x2060)

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

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

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

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

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

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

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

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

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

5.6.2.19 **#define ETRAC8(fmt, arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8) printf("%s> ETRC: " fmt "\n", __FUNCTION__, arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8);**

5.6.2.20 **#define ETRAC9(fmt, arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9) printf("%s> ETRC: " fmt "\n", __FUNCTION__, arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9);**

5.6.3 Enumeration Type Documentation

5.6.3.1 enum ecmdFormatType_t

This is the different formats in which the output file will be written.

Enumeration values:

ECMD_SAVE_FORMAT_BINARY binary file with header with info like bit length etc

ECMD_SAVE_FORMAT_BINARY_DATA binary file with data only

ECMD_SAVE_FORMAT_ASCII ascii text file with header having same info like binary hdr

ECMD_SAVE_FORMAT_XSTATE xstate text file with header having same info like binary hdr

5.7 ecmdReturnCodes.H File Reference

All Return Codes for the eCmd Capi.

Defines

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

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

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

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

  ecmdQueryFileLocation was unable to find the file you requested

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

  Not enough arguments provided to the function.

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

  getSpyEnum or putSpyEnum used an invalid enum

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

  getSpy or getSpyEnum failed with invalid ECC detected in the hardware

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

  getSpyEnum or putSpyEnum was called on a non-enumerated spy

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

  getSpy or Putspy was called on an edial

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

  Spy functions found an invalid Spy name or type.

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

  Too much data was provided to a write function.

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

  Too little data was provided to a write function.

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

  Invalid ring name was provided.

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

  Invalid array name was provided.

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

  There was an error processing the configuration information.

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

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

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

  JTag function called on non-jtag attached chip.

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

  Fsi function called on non-fsi attached chip.

- #define **ECMD_INVALID_SPR** (ECMD_ERR_ECMD | 0x1034)
  Invalid SPR was specified to get/put spr functions.
- #define **ECMD_INVALID_GPR** (ECMD_ERR_ECMD | 0x1035)
  Invalid GPR number was specified to get/put gpr functions.
- #define **ECMD_INVALID_FPR** (ECMD_ERR_ECMD | 0x1036)
  Invalid FPR number was specified to get/put fpr functions.
- #define **ECMD_RING_CACHE_ENABLED** (ECMD_ERR_ECMD | 0x1037)
  Ring Cache enabled during call non-cache enabled function.
- #define **ECMD_INVALID_CONFIG_NAME** (ECMD_ERR_ECMD | 0x1038)
  An Invalid name was used to set/get a configuation setting.
- #define **ECMD_SPY_GROUP_MISMATCH** (ECMD_ERR_ECMD | 0x1039)
  A mismatch was found reading a group spy not all groups set the same.
- #define **ECMD_INVALID_CLOCK_DOMAIN** (ECMD_ERR_ECMD | 0x1040)
  An invalid clock domain name was specified.
- #define **ECMD_CLOCKS_ALREADY_OFF** (ECMD_ERR_ECMD | 0x1041)
  A stopclocks was requested when clocks are already off.
- #define **ECMD_CLOCKS_ALREADY_ON** (ECMD_ERR_ECMD | 0x1042)
  A startclocks was requested when clocks are already on.
- #define **ECMD_UNABLE_TO_OPEN_SCANDEF** (ECMD_ERR_ECMD | 0x1043)
  eCMD was unable to open the scandef
- #define **ECMD_INVALID_LATCHNAME** (ECMD_ERR_ECMD | 0x1044)
  eCMD was unable to find the specified latch in the scandef
- #define **ECMD_POLLING_FAILURE** (ECMD_ERR_ECMD | 0x1045)
  eCMD failed waiting for a poll to match expected value
- #define **ECMD_TARGET_INVALID_TYPE** (ECMD_ERR_ECMD | 0x1046)
  Target specified an object that was inappropriate for the function.
- #define **ECMD_EXTENSION_NOT_SUPPORTED** (ECMD_ERR_ECMD | 0x1047)
  The current plugin does not supported the requested extension.
- #define **ECMD_ISTEPS_INVALID_STEP** (ECMD_ERR_ECMD | 0x1048)
  An invalid step name was provided.
- #define **ECMD_UNABLE_TO_OPEN_SCANDEFHASH** (ECMD_ERR_-ECMD | 0x1049)
  eCMD was unable to open the scandefhash

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

  Multiple ring keys matching the same latchname found.

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

  Command interpreter didn't understand command.

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

  An expect was performed and a miscompare was found.

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

  An Error occurred trying to process the scandef file.

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

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

- #define **ECMD_DBUF_SUCCESS** 0x0

  DataBuffer returned successfully.

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

  Initialization of the DataBuffer failed.

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

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

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

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

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

  Function not included in this version of DataBuffer.

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

  Args provided to dataBuffer were invalid.

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

  String data didn't match expected input format.

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

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

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

  In readFile specified format not found in the data file.

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

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

5.7.1 Detailed Description

All Return Codes for the eCmd Capi.

5.7.2 Define Documentation

5.7.2.1 #define ECMD_ERR_UNKNOWN 0x00000000

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

5.7.2.2 #define ECMD_ERR_ECMD 0x01000000

Error came from eCMD.

5.7.2.3 #define ECMD_ERR_CRONUS 0x02000000

Error came from Cronus.

5.7.2.4 #define ECMD_ERR_IP 0x04000000

Error came from IP GFW.

5.7.2.5 #define ECMD_ERR_Z 0x08000000

Error came from Z GFW.

5.7.2.6 #define ECMD_SUCCESS 0x0

API Returned Successfully.

5.7.2.7 #define ECMD_INVALID_DLL_VERSION (ECMD_ERR_ECMD | 0x1000)

Dll Version didn't match the Client version detected.

5.7.2.8 #define ECMD_INVALID_DLL_FILENAME (ECMD_ERR_ECMD | 0x1001)

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

5.7.2.9 #define ECMD_DLL_LOAD_FAILURE (ECMD_ERR_ECMD | 0x1002)

Error occured on call to dlopen.

5.7.2.10 #define ECMD_DLL_UNLOAD_FAILURE (ECMD_ERR_ECMD | 0x1003)

Error occurred on call to dlclose.

5.7.2.11 #define ECMD_DLL_UNINITIALIZED (ECMD_ERR_ECMD | 0x1004)

A function was called before ecmdLoadDll was called.

5.7.2.12 #define ECMD_DLL_INVALID (ECMD_ERR_ECMD | 0x1005)

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

5.7.2.13 #define ECMD_FAILURE (ECMD_ERR_ECMD | 0x1010)

General Failure occurred in eCMD.

5.7.2.14 #define ECMD_TARGET_NOT_CONFIGURED (ECMD_ERR_ECMD | 0x1011)

Chip target provided was not configured in the system.

5.7.2.15 #define ECMD_FUNCTION_NOT_SUPPORTED (ECMD_ERR_ECMD | 0x1012)

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

5.7.2.16 #define ECMD_UNKNOWN_FILE (ECMD_ERR_ECMD | 0x1013)

ecmdQueryFileLocation was unable to find the file you requested

5.7.2.17 #define ECMD_INVALID_ARGS (ECMD_ERR_ECMD | 0x1020)

Not enough arguments provided to the function.

5.7.2.18 #define ECMD_INVALID_SPY_ENUM (ECMD_ERR_ECMD | 0x1021)

getSpyEnum or putSpyEnum used an invalid enum

5.7.2.19 #define ECMD_SPY_FAILED_ECC_CHECK (ECMD_ERR_ECMD | 0x1022)

getSpy or getSpyEnum failed with invalid ECC detected in the hardware

5.7.2.20 #define ECMD_SPY_NOT_ENUMERATED (ECMD_ERR_ECMD | 0x1023)

getSpyEnum or putSpyEnum was called on a non-enumerated spy

5.7.2.21 #define ECMD_SPY_IS_EDIAL (ECMD_ERR_ECMD | 0x1024)

getSpy or Putspy was called on an edial

5.7.2.22 #define ECMD_INVALID_SPY (ECMD_ERR_ECMD | 0x1025)

Spy functions found an invalid Spy name or type.

5.7.2.23 #define ECMD_DATA_OVERFLOW (ECMD_ERR_ECMD | 0x1026)

Too much data was provided to a write function.

5.7.2.24 #define ECMD_DATA_UNDERFLOW (ECMD_ERR_ECMD | 0x1027)

Too little data was provided to a write function.

5.7.2.25 #define ECMD_INVALID_RING (ECMD_ERR_ECMD | 0x1028)

Invalid ring name was provided.

5.7.2.26 #define ECMD_INVALID_ARRAY (ECMD_ERR_ECMD | 0x1029)

Invalid array name was provided.

5.7.2.27 #define ECMD_INVALID_CONFIG (ECMD_ERR_ECMD | 0x1030)

There was an error processing the configuration information.

5.7.2.28 #define ECMD_CLOCKS_IN_INVALID_STATE (ECMD_ERR_ECMD | 0x1031)

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

5.7.2.29 #define ECMD_NON_JTAG_CHIP (ECMD_ERR_ECMD | 0x1032)

JTag function called on non-jtag attached chip.

5.7.2.30 #define ECMD_NON_FSI_CHIP (ECMD_ERR_ECMD | 0x1033)

Fsi function called on non-fsi attached chip.

5.7.2.31 #define ECMD_INVALID_SPR (ECMD_ERR_ECMD | 0x1034)

Invalid SPR was specified to get/put spr functions.

5.7.2.32 #define ECMD_INVALID_GPR (ECMD_ERR_ECMD | 0x1035)

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

5.7.2.33 #define ECMD_INVALID_FPR (ECMD_ERR_ECMD | 0x1036)

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

5.7.2.34 #define ECMD_RING_CACHE_ENABLED (ECMD_ERR_ECMD | 0x1037)

Ring Cache enabled during call non-cache enabled function.

5.7.2.35 #define ECMD_INVALID_CONFIG_NAME (ECMD_ERR_ECMD | 0x1038)

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

5.7.2.36 #define ECMD_SPY_GROUP_MISMATCH (ECMD_ERR_ECMD | 0x1039)

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

5.7.2.37 #define ECMD_INVALID_CLOCK_DOMAIN (ECMD_ERR_ECMD | 0x1040)

An invalid clock domain name was specified.

5.7.2.38 #define ECMD_CLOCKS_ALREADY_OFF (ECMD_ERR_ECMD | 0x1041)

A stopclocks was requested when clocks are already off.

5.7.2.39 #define ECMD_CLOCKS_ALREADY_ON (ECMD_ERR_ECMD | 0x1042)

A startclocks was requested when clocks are already on.

5.7.2.40 #define ECMD_UNABLE_TO_OPEN_SCANDEF (ECMD_ERR_ECMD | 0x1043)

eCMD was unable to open the scandef

5.7.2.41 #define ECMD_INVALID_LATCHNAME (ECMD_ERR_ECMD | 0x1044)

eCMD was unable to find the specified latch in the scandef

5.7.2.42 #define ECMD_POLLING_FAILURE (ECMD_ERR_ECMD | 0x1045)

eCMD failed waiting for a poll to match expected value

5.7.2.43 #define ECMD_TARGET_INVALID_TYPE (ECMD_ERR_ECMD | 0x1046)

Target specified an object that was inappropriate for the function.

5.7.2.44 #define ECMD_EXTENSION_NOT_SUPPORTED (ECMD_ERR_ECMD | 0x1047)

The current plugin does not supported the requested extension.

5.7.2.45 #define ECMD_ISTEPS_INVALID_STEP (ECMD_ERR_ECMD | 0x1048)

An invalid step name was provided.

5.7.2.46 #define ECMD_UNABLE_TO_OPEN_SCANDEFHASH (ECMD_ERR_ECMD | 0x1049)

eCMD was unable to open the scandefhash

5.7.2.47 #define ECMD_SCANDEFHASH_MULT_RINGS (ECMD_ERR_ECMD | 0x1050)

Multiple ring keys matching the same latchname found.

5.7.2.48 #define ECMD_INT_UNKNOWN_COMMAND (ECMD_ERR_ECMD | 0x1900)

Command interpreter didn't understand command.

5.7.2.49 #define ECMD_EXPECT_FAILURE (ECMD_ERR_ECMD | 0x1901)

An expect was performed and a miscompare was found.

5.7.2.50 #define ECMD_SCANDEF_LOOKUP_FAILURE (ECMD_ERR_ECMD | 0x1902)

An Error occurred trying to process the scandef file.

5.7.2.51 #define ECMD_DATA_BOUNDS_OVERFLOW (ECMD_ERR_ECMD | 0x1903)

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

5.7.2.52 #define ECMD_DBUF_SUCCESS 0x0

DataBuffer returned successfully.

5.7.2.53 #define ECMD_DBUF_INIT_FAIL (ECMD_ERR_ECMD | 0x2000)

Initialization of the DataBuffer failed.

5.7.2.54 #define ECMD_DBUF_BUFFER_OVERFLOW (ECMD_ERR_ECMD | 0x2010)

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

5.7.2.55 #define ECMD_DBUF_XSTATE_ERROR (ECMD_ERR_ECMD | 0x2020)

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

5.7.2.56 #define ECMD_DBUF_UNDEFINED_FUNCTION (ECMD_ERR_ECMD | 0x2030)

Function not included in this version of DataBuffer.

5.7.2.57 #define ECMD_DBUF_INVALID_ARGS (ECMD_ERR_ECMD | 0x2040)

Args provided to dataBuffer were invalid.

5.7.2.58 #define ECMD_DBUF_INVALID_DATA_FORMAT (ECMD_ERR_ECMD | 0x2041)

String data didn't match expected input format.

5.7.2.59 #define ECMD_DBUF_FOPEN_FAIL (ECMD_ERR_ECMD | 0x2050)

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

5.7.2.60 #define ECMD_DBUF_FILE_FORMAT_MISMATCH (ECMD_ERR_ECMD | 0x2051)

In readFile specified format not found in the data file.

5.7.2.61 #define ECMD_DBUF_NOT_OWNER (ECMD_ERR_ECMD | 0x2060)

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

5.8 ecmdSharedUtils.H File Reference

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

```
#include <string>
#include <vector>
#include <inttypes.h>
#include <ecmdDataBuffer.H>
```

Command Line Parsing Functions

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

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

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

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

- void **ecmdParseTokens** (std::string line, const char *seperators, std::vector< std::string > &tokens)

  Breaks the string line into tokens based on all chars in seperators.

- std::string **ecmdGenEbcdic** (**ecmdDataBuffer** &i_data, int start, int bitLen)

  Turns the data in the buffer into ebcdic text.

Functions

- uint32_t **ecmdHexToUInt32** (const char *i_str)

  Converts strings to unsigned int values. The input format is 0xABCDEF.

- uint32_t **ecmdHashString32** (const char *i_str, uint32_t i_c)

  Calculates a 32bit hash value for a given string.

5.8.1 Detailed Description

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

5.8.2 Function Documentation

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

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

Return values:
1 if option found, 0 otherwise

Parameters:

io_argc Pointer to number of elements in io_argv array

io_argv Array of strings passed in from command line

i_option Option to look for

See also:

ecmdParseOptionWithArgs(p. 173)

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

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

Return values:

Value of option arg if found, NULL otherwise

Parameters:

io_argc Pointer to number of elements in io_argv array

io_argv Array of strings passed in from command line

i_option Option to look for

See also:

ecmdParseOptionWithArgs(p. 173)

5.8.2.3 void ecmdParseTokens (std::string *line*, const char * *seperators*, std::vector< std::string > & *tokens*)

Breaks the string line into tokens based on all chars in seperators.

Parameters:

line String to tokenize

seperators String of characters to use as seperators

tokens Vector of strings that contain all the tokens

5.8.2.4 std::string ecmdGenEbcdic (ecmdDataBuffer & *i_data*, int *start*, int *bitLen*)

Turns the data in the buffer into ebcdic text.

Parameters:

i_data Data to convert

start Bit to start at

bitLen Number of bits

5.8.2.5 uint32_t ecmdHexToUInt32 (const char * *i_str*)

Converts strings to unsigned int values. The input format is 0xABCDEF.

Parameters:
i_str String in hexadecimal notation

Date:
Tue Sep 21 13:22:33 2004

Return values:
uint32_t value of converted input string

5.8.2.6 uint32_t ecmdHashString32 (const char * *i_str*, uint32_t *i_c*)

Calculates a 32bit hash value for a given string.

LICENSE: By Bob Jenkins, 1996. `bob_jenkins@burtleburtle.net`. You may use this code any way you wish, private, educational, or commercial. It's free. See `http://burtleburtle.net/bob/hash/doobs.html`

Parameters:
i_str String to convert to hash

i_c Start value for hash.

Return values:
Hash value

5.9 ecmdStructs.H File Reference

All the Structures required for the eCMD Capi.

```
#include <inttypes.h>
#include <list>
#include <vector>
#include <string>
#include <ecmdDataBuffer.H>
```

Classes

- struct **ecmdDllInfo**

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

- struct **ecmdChipTarget**

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

- struct **ecmdThreadData**

  Used for the ecmdQueryConfig function to return thread data.

- struct **ecmdCoreData**

  Used for the ecmdQueryConfig function to return core data.

- struct **ecmdChipData**

  Used for the ecmdQueryConfig function to return chip data.

- struct **ecmdSlotData**

  Used for the ecmdQueryConfig function to return slot data.

- struct **ecmdNodeData**

  Used for the ecmdQueryConfig function to return node data.

- struct **ecmdCageData**

  Used for the ecmdQueryConfig function to return cage data.

- struct **ecmdQueryData**

  Used by the ecmdQueryConfig function to return data.

- struct **ecmdRingData**

  Used for the ecmdQueryRing function to return ring info.

- struct **ecmdArrayData**

  Used for the ecmdQueryArray function to return array info.

- struct **ecmdArrayEntry**

  Used by the getArrayMultiple function to pass data.

- struct **ecmdSpyGroupData**

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

- struct **ecmdNameEntry**

  Used by get/putSprMultiple function to pass data.

- struct **ecmdNameVectorEntry**

  Used by getTraceArrayMultiple function to pass data.

- struct **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** "4.1d"

  eCMD API Version

- #define **QD_HDR_MAGIC** 0xFFFFFFF1
- #define **CAGE_HDR_MAGIC** 0xFFFFFF2F
- #define **NODE_HDR_MAGIC** 0xFFFFF3FF
- #define **SLOT_HDR_MAGIC** 0xFFFF4FFF
- #define **CHIP_HDR_MAGIC** 0xFFF5FFFF
- #define **CORE_HDR_MAGIC** 0xFF6FFFFF
- #define **THREAD_HDR_MAGIC** 0xF7FFFFFF
- #define **ECMD_CHIPT_PROCESSOR** "pu"
- #define **ECMD_CHIPT_MEM_BUF** "memb"
- #define **ECMD_CHIPT_MEM_CNTRL** "memc"
- #define **ECMD_CHIPT_MEM_L2CACHE** "l2cache"
- #define **ECMD_CHIPT_MEM_L3CACHE** "l3cache"
- #define **ECMD_CHIPT_IOBDG** "iobdg"
- #define **ECMD_CHIPT_IOHUB** "iohub"
- #define **ECMD_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. 19) *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. 16) *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.

- enum **ecmdFileType_t** {

  ECMD_FILE_SCANDEF, **ECMD_FILE_SPYDEF**, **ECMD_FILE_ARRAYDEF**, **ECMD_FILE_HELPTEXT**,

  ECMD_FILE_SCOMDATA, **ECMD_FILE_SPYDEFHASH**, **ECMD_FILE_SCANDEFHASH** }

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

- enum **ecmdConfigLoopType_t** {

  ECMD_SELECTED_TARGETS_LOOP, **ECMD_SELECTED_TARGETS_LOOP_DEFALL**, **ECMD_SELECTED_TARGETS_LOOP_VD**, **ECMD_SELECTED_TARGETS_LOOP_VD_DEFALL**,

  ECMD_ALL_TARGETS_LOOP }

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

- enum **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.

- enum **efppInOut_t** {

  ECMD_FPP_RETOUT, **ECMD_FPP_FUNCTIONIN**, **ECMD_FPP_FUNCTIONOUT**, **ECMD_FPP_JUSTIN**,

  ECMD_FPP_JUSTOUT }

  Used by the eCMD Function parm trace printer.

- enum **ecmdConfigValid_t** { **ECMD_CONFIG_VALID_FIELD_NONE**, **ECMD_CONFIG_VALID_FIELD_ALPHA**, **ECMD_CONFIG_VALID_FIELD_NUMERIC**, **ECMD_CONFIG_VALID_FIELD_BOTH** }

  Used by the get/set configuration functions to specify what data is good.

5.9.1 Detailed Description

All the Structures required for the eCMD Capi.

5.9.2 Define Documentation

5.9.2.1 #define ECMD_CAPI_VERSION "4.1d"

eCMD API Version

5.9.2.2 #define QD_HDR_MAGIC 0xFFFFFFF1

5.9.2.3 #define CAGE_HDR_MAGIC 0xFFFFFF2F

5.9.2.4 #define NODE_HDR_MAGIC 0xFFFFF3FF

5.9.2.5 #define SLOT_HDR_MAGIC 0xFFFF4FFF

5.9.2.6 #define CHIP_HDR_MAGIC 0xFFF5FFFF

5.9.2.7 #define CORE_HDR_MAGIC 0xFF6FFFFF

5.9.2.8 #define THREAD_HDR_MAGIC 0xF7FFFFFF

5.9.2.9 #define ECMD_CHIPT_PROCESSOR "pu"

Predefined common chip names for **ecmdChipData.chipCommonType**(p. 17)

5.9.2.10 #define ECMD_CHIPT_MEM_BUF "memb"

5.9.2.11 #define ECMD_CHIPT_MEM_CNTRL "memc"

5.9.2.12 #define ECMD_CHIPT_MEM_L2CACHE "l2cache"

5.9.2.13 #define ECMD_CHIPT_MEM_L3CACHE "l3cache"

5.9.2.14 #define ECMD_CHIPT_IOBDG "iobdg"

5.9.2.15 #define ECMD_CHIPT_IOHUB "iohub"

5.9.2.16 #define ECMD_CHIPFLAG_BUSMASK 0xC0000000

Defines for the **ecmdChipData**(p. 16) chipFlags field

5.9.2.17 #define ECMD_CHIPFLAG_RSVDBUS1 0x00000000

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

5.9.2.18 #define ECMD_CHIPFLAG_JTAG 0x40000000

5.9.2.19 #define ECMD_CHIPFLAG_FSI 0x80000000

5.9.2.20 #define ECMD_CHIPFLAG_RSVDBUS2 0xC0000000

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

5.9.3 Enumeration Type Documentation

5.9.3.1 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.

5.9.3.2 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.

5.9.3.3 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.

5.9.3.4 enum ecmdChipTargetState_t

Used by **ecmdChipTarget**(p. 19) 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.

5.9.3.5 enum ecmdChipInterfaceType_t

Used in **ecmdChipData**(p. 16) to describe the interface macro used by the chip.

Enumeration values:

ECMD_INTERFACE_ACCESS Standard Jtag Access Macro.

ECMD_INTERFACE_CFAM CommonFirmwareAccessMacro.

ECMD_INTERFACE_UNKNOWN Unknown Interface.

5.9.3.6 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.

5.9.3.7 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.

5.9.3.8 enum ecmdSpyType_t

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

See also:

ecmdSpyData(p. 83)

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.

5.9.3.9 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

ECMD_FILE_SPYDEFHASH Hash file for spy definition.

ECMD_FILE_SCANDEFHASH Hash file for the scandef.

5.9.3.10 enum ecmdConfigLoopType_t

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

Enumeration values:

ECMD_SELECTED_TARGETS_LOOP Loop on only targets in the system the user specified with -p# -c# -n#, etc. if not specified default to 0.

ECMD_SELECTED_TARGETS_LOOP_DEFALL Loop on only targets in the system the user specified with -p# -c# -n#, etc. if not specified default to all.

ECMD_SELECTED_TARGETS_LOOP_VD Loop on only targets in the system (Variable Depth) only to depth user specified (ie -n0 then -s and below are unused) if not specified default to 0.

ECMD_SELECTED_TARGETS_LOOP_VD_DEFALL Loop on only targets in the system (Variable Depth) only to depth user specified (ie -n0 then -s and below are unused) if not specified default to all.

ECMD_ALL_TARGETS_LOOP Loop on all valid targets in the system.

5.9.3.11 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.

5.9.3.12 enum ecmdTraceType_t

Used by ecmdSetTraceMode to specify which trace to control.

Enumeration values:

ECMD_TRACE_SCAN Scan Trace.

ECMD_TRACE_PROCEDURE Procedure Trace.

5.9.3.13 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.

5.9.3.14 enum efppInOut_t

Used by the eCMD Function parm trace printer.

Enumeration values:

ECMD_FPP_RETOUT Designates the failing return code out of the api function.

ECMD_FPP_FUNCTIONIN Designates the call in of the api function.

ECMD_FPP_FUNCTIONOUT Designatest the call out of the api function.

ECMD_FPP_JUSTIN Designatest the call in of the api function in debug 8 mode.

ECMD_FPP_JUSTOUT Designatest the call out of the api function in debug 8 mode and rc==0.

5.9.3.15 enum ecmdConfigValid_t

Used by the get/set configuration functions to specify what data is good.

Enumeration values:

ECMD_CONFIG_VALID_FIELD_NONE No field is valid, must have been an error.

ECMD_CONFIG_VALID_FIELD_ALPHA The string field contains valid data.

ECMD_CONFIG_VALID_FIELD_NUMERIC The numeric field contains valid data.

ECMD_CONFIG_VALID_FIELD_BOTH Bothe the string and numeric fields contain valid data.

5.9.4 Function Documentation

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

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

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

Used to sort Node entries in an **ecmdNodeData**(p. 74) list.

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

Used to sort Slot entries in an **ecmdSlotData**(p. 81) list.

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

Used to sort Chip entries (based on Pos) in an **ecmdChipData**(p. 16) list.

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

Used to sort Core entries in an **ecmdCoreData**(p. 23) list.

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

Used to sort Thread entries in an **ecmdThreadData**(p. 87) list.

5.9.4.7 std::string ecmdGetSharedLibVersion ()

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

5.10 ecmdUtils.H File Reference

Useful functions for use throughout the ecmd C API.

```
#include <inttypes.h>
#include <string>
#include <vector>
#include <ecmdClientCapi.H>
```

Defines

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

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.

- uint32_t **decToUInt32** (const char *i_decstr)

  Converts decimal string to uint32_t.

- std::string **ecmdWriteDataFormatted** (**ecmdDataBuffer** &i_data, std::string i_format, uint64_t i_address=0)

  Formats data from data buffer into a string according to format flag and returns the string.

- std::string **ecmdBitsHeader** (int i_initCharOffset, int i_blockSize, int i_numCols, int i_maxBitWidth)

  Print the bits header used in the output formats.

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

  Returns a formatted string containing the data in the given **ecmdChipTarget**(p. 19).

- 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.

- void **ecmdFunctionParmPrinter** (**efppInOut_t** inOut, const char *fprototypeStr, std::vector< void * > args)

  Print the parameters pass to a given function.

5.10.1 Detailed Description

Useful functions for use throughout the ecmd C API.

5.10.2 Define Documentation

5.10.2.1 #define PTRAC0(fmt)

Value:

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

5.10.2.2 #define PTRAC1(fmt, arg1)

Value:

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

5.10.2.3 #define PTRAC2(fmt, arg1, arg2)

Value:

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

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

Value:

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

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

Value:

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

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

Value:

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

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

Value:

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

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

Value:

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

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

Value:

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

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

Value:

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

5.10.3 Function Documentation

5.10.3.1 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. 19) 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. 188)

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

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

Parameters:

io_target **ecmdChipTarget**(p. 19) that contains info about next target to process

io_state Used internally to keep track of state, must be passed from output of ecmdConfig-LooperInit

Return values:

1 if io_target is valid, 0 if it is not

See also:

ecmdConfigLooperInit(p. 188)

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

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

Return values:

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

Parameters:

o_data **ecmdDataBuffer**(p. 25) 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

5.10.3.4 uint32_t decToUInt32 (const char * *i_decstr*)

Converts decimal string to uint32_t.

Return values:
uint32_t value of converted input string

Parameters:
i_decstr string of characters containing data

5.10.3.5 std::string ecmdWriteDataFormatted (ecmdDataBuffer & *i_data*, std::string *i_format*, uint64_t *i_address* = 0)

Formats data from data buffer into a string according to format flag and returns the string.

Returns:
String of formatted data

Parameters:
i_data **ecmdDataBuffer**(p. 25) where data to format is stored
i_format Flag that tells how to parse the data into a string, e.g., "b" = binary, "x" = hex left
i_address A base address value that can be used in formating certain data- i.e., data from memory

5.10.3.6 std::string ecmdBitsHeader (int *i_initCharOffset*, int *i_blockSize*, int *i_numCols*, int *i_maxBitWidth*)

Print the bits header used in the output formats.

Parameters:
i_initCharOffset char offset on screen to start printing
i_blockSize Binary block size (ie. column char size)
i_numCols Number of columns to display
i_maxBitWidth Maximum number of bits to display - this is actual data valid so we don't display more columns then we need

Returns:
String of formatted data

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

Returns a formatted string containing the data in the given **ecmdChipTarget**(p. 19).

Returns:
String with formatted target data

Parameters:
i_target **ecmdChipTarget**(p. 19) containing data to format into string

5.10.3.8 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. 19) that information is requested for

o_data **ecmdChipData**(p. 16) struct that contains detailed info on chip ec level, etc.

5.10.3.9 uint32_t ecmdDisplayDllInfo ()

Function calls ecmdQueryDllInfo and displays the output to stdout.

Return values:
ECMD_SUCCESS if successful

nonzero on failure

5.10.3.10 void ecmdFunctionParmPrinter (efppInOut_t *inOut*, const char * *fprototypeStr*, std::vector< void * > *args*)

Print the parameters pass to a given function.

Return values:
Void return

Parameters:
inOut Tell Parm Printer if we are going into or out of the function

fprototypeStr function prototype

args function argument vector

Index