NAME

LDPCFecSession −

SYNOPSIS

```
#include <ldpc_fec.h>
```

Inherited by **LDPCFecScheme**.

Public Member Functions

LDPCFecSession ()
˜LDPCFecSession ()
ldpc_error_status InitSession (int nbSourceSymbols, int nbParitySymbols, int symbolSize, int flags=FLAG_BOTH, int **seed**=1, **SessionType** codecType=TypeTRIANGLE, int leftDegree=3)
ldpc_error_status SetCallbackFunctions (void *(*DecodedSymbol_callback)(void *context, int size, int symbol_seqno), void *(*AllocTmpBuffer_callback)(void *context, int size), void *(*GetData_callback)(void *context, void *symbol), void *(*GetDataPtrOnly_callback)(void *context, void *symbol), **ldpc_error_status**(*StoreData_callback)(void *context, void *symbol), **ldpc_error_status**(*FreeSymbol_callback)(void *context, void *symbol), void *context_4_callback=NULL)
void **EndSession** ()
bool **IsInitialized** ()
void **SetVerbosity** (int verb)
void **MoreAbout** (FILE *out)
int **GetMaxN** ()
ldpc_error_status BuildParitySymbol (void *symbol_canvas[], int paritySymbol_index, void *paritySymbol)
ldpc_error_status DecodingStepWithSymbol (void *symbol_canvas[], void *new_symbol, int new_symbol_seqno, bool store_symbol)
ldpc_error_status DecodingStepWithSymbol (void *symbol_canvas[], void *new_symbol, int new_symbol_seqno)
bool **SymbolAlreadyKnown** (void *symbol_canvas[], int new_symbol_seqno)
bool **IsDecodingComplete** (void *symbol_canvas[])
unsigned int **GetNbXor** (void)
void **ResetNbXor** (void)

Detailed Description

This is the LDPC FEC session class, where all the context information is kept for encoding/decoding this block. To 'k' source symbols, the LDPC codec can add 'n-k' parity (or FEC) symbols, for a total of 'n' symbols. Source symbols are numbered {0; k-1} and parity symbols {k; n-1}. There must be one such FEC session instance per FEC block.

When LDPCFecSession and **LDPCFecScheme** are both used, the LDPCFecSession MUST be initialized first (with **InitSession()**), THEN the **LDPCFecScheme** (with InitScheme()).

WARNING: the following class contains a lot of checking code that is only available in DEBUG mode (set -DDEBUG on the compiling line). Whenever used with a new application, first validate your code in DEBUG mode, and switch to production code only in a second step...

Constructor & Destructor Documentation

LDPCFecSession::LDPCFecSession ()

LDPCFecSession Contructor and Destructor.

LDPCFecSession::˜LDPCFecSession ()

Member Function Documentation

ldpc_error_status LDPCFecSession::InitSession (int nbSourceSymbols, int nbParitySymbols, int symbolSize, int flags = `FLAG_BOTH`, int seed = `1`, SessionType codecType = `TypeTRIANGLE`, int leftDegree = `3`)

InitSession: Initializes the LDPC session.

Parameters:

nbSourceSymbols (IN) number of source symbols (i.e. k).
nbParitySymbols (IN) number of parity symbols (i.e. n-k). Be careful that n-k cannot be less than the left degree (i.e. 3 by default), otherwise an error is returned.
symbolSize (IN) symbol size in bytes. It does NOT need to be multiple of 4, any value is accepted.
flags (IN) session flags (FLAG_CODER, FLAG_DECODER, ...).
seed (IN) seed used to build the parity check matrix (H).
codecType (IN) Type of codec algorithm and matrix to use. Can be on of TypeLDGM, TypeSTAIRS, or TypeTRIANGLE.
leftDegree (IN) number of equations in which a symbol is involved. 3 (default) is the optimal value for TypeSTAIRS and TypeTRIANGLE codes, DO NOT change. With TypeLDGM, higher values are usually preferable (see INRIA Research Report 5225, June 2004).

Returns:

Completion status (LDPC_OK or LDPC_ERROR).

ldpc_error_status LDPCFecSession::SetCallbackFunctions (void *(*)(void *context, int size, int symbol_seqno) DecodedSymbol_callback, void *(*)(void *context, int size) AllocTmpBuffer_callback, void *(*)(void *context, void *symbol) GetData_callback, void *(*)(void *context, void *symbol) GetDataPtrOnly_callback, ldpc_error_status(*)(void *context, void *symbol) StoreData_callback, ldpc_error_status(*)(void *context, void *symbol) FreeSymbol_callback, void * context_4_callback = `NULL`**)**

SetCallbackFunctions: Set the various callback functions for this session.

- The DecodedSymbol callback function is called each time a source symbol is decoded by the **DecodingStepWithSymbol()** function. What this function does is application-dependant, but it must return a pointer to a data buffer, left uninitialized, of the appropriate size. In EXTERNAL_MEMORY_MGMT_SUPPORT mode, this function returns an opaque symbol pointer. The associated buffer, where actual data will be stored, must be retrieved via the GetData callback.

In EXTERNAL_MEMORY_MGMT_SUPPORT mode, the following callbacks are defined:

- The AllocTmpBuffer callback is called each time a temporary buffer is required by the system, e.g. to store a partial sum (check node). This function returns a symbol pointer, and accessing the data buffer requires a call to the GetData callback. The associated data buffer MUST be initialized to ’0’ by the callback.
- The GetData callback is called each time the data associated to a symbol must be read. What this function does is application-dependant.
- The StoreData callback is called each time a symbol’s buffer has been updated and must be stored reliably by the memory mgmt system. What this function does is application-dependant.
- The FreeSymbol callback is called each time a symbol (or temporary buffer) is no longer required and can be free’d by the memory mgmt system.

All callback functions require an opaque context parameter, that is the same parameter as the one given to **DecodingStepWithSymbol()**.

Parameters:

DecodedSymbol_callback (IN) Pointer to an application’s callback. Given the size of a newly created source symbol and its sequence number, this function enables the callee to allocate a symbol structure. This function returns a pointer to the data buffer allocated or to the symbol in EXTERNAL_MEMORY_MGMT_SUPPORT mode. This callback is never called when decoding a parity symbol!
AllocTmpBuffer_callback (IN) Pointer to an application’s callback. Valid in EXTERNAL_MEMORY_MGMT_SUPPORT mode. Given the desired buffer size, this function allocates a symbol that will contain a buffer of appropriate size and initialized to ’0’.

GetData_callback (IN) Pointer to an application's callback. Valid in EXTERNAL_MEMORY_MGMT_SUPPORT mode. Given the symbol pointer, this function returns the data buffer, after making sure that this latter is available and up-to-date.
GetDataPtrOnly_callback (IN) Pointer to an application's callback. Valid in EXTERNAL_MEMORY_MGMT_SUPPORT mode. Same as GetData_callback, except that no check is made to make sure data is available and up-to-date. It makes sense when buffer has just been allocated before, for instance because this is a destination buffer in a memcpy() syscall.
StoreData_callback (IN) Pointer to an application's callback. Valid in EXTERNAL_MEMORY_MGMT_SUPPORT mode. Given the symbol pointer, this function stores data reliably in the memory mgmt system.
FreeSymbol_callback (IN) Pointer to an application's callback. Valid in EXTERNAL_MEMORY_MGMT_SUPPORT mode. This function will be called with a symbol pointer, so that the external memory mgmt system can free the associated buffer.
context_4_callback (IN) Pointer to context that will be passed to the callback function (if any). This context is not interpreted by this function.

Returns:
Completion status (LDPC_OK or LDPC_ERROR).

void LDPCFecSession::EndSession ()
EndSession: Ends the LDPC session, cleans up everything.

bool LDPCFecSession::IsInitialized () `[inline]`
IsInitialized: Check if the LDPC session has been initialized.

Returns:
TRUE if the session is ready and initialized, FALSE if not.

void LDPCFecSession::SetVerbosity (int verb)
Set the verbosity level.

Parameters:
verb (IN) new verbosity level (0: no trace, 1: all traces)

void LDPCFecSession::MoreAbout (FILE * out)
Prints version number and copyright information about this codec.

Parameters:
out (IN) FILE handle where the string should be written.

int LDPCFecSession::GetMaxN () `[inline]`
Returns the maximum encoding block length (n parameter). This limit is not LDPC-* specifi c that are nature large bloc FEC codes, meaning that (k,n) can both be very very large. This is a codec specifi c limit, due to the way the codec is implemented. See **ldpc_profile.h**: If SPARSE_MATRIX_OPT_SMALL_INDEX is defi ned, then k <= n < 2ˆ15; Else k <= n < 2ˆ31 The limits are essentially over the n parameter, but given the desired FEC Expansion ratio n/k (or the code rate, k/n), it will also limit the source block length (k parameter).

Returns:
Maximum n value (A.K.A. encoding block length).

ldpc_error_status LDPCFecSession::BuildParitySymbol (void * symbol_canvas[], int paritySymbol_index, void * paritySymbol)
Build a new parity symbol.

Parameters:

symbol_canvas (IN) Array of source and parity symbols. This is a table of n pointers to buffers containing the source and parity symbols.
paritySymbol_index (IN) Index of parity symbol to build in {0.. n-k-1} range (!)
paritySymbol (IN-OUT) Pointer to the parity symbol buffer that will be built. This buffer MUST BE allocated before, but NOT cleared (memset(0)) since this function will do it.

Returns:

Completion status (LDPC_OK or LDPC_ERROR).

ldpc_error_status LDPCFecSession::DecodingStepWithSymbol (void * symbol_canvas[], void * new_symbol, int new_symbol_seqno, bool store_symbol)

Perform a new decoding step thanks to the newly received symbol.

Parameters:

symbol_canvas (IN-OUT) Global array of received or rebuilt source symbols (parity symbols need not be stored here). This is a table of k pointers to buffers. This array must be cleared (memset(0)) upon the fi rst call to this function. It will be automatically updated, with pointers to symbols received or decoded, by this function.
new_symbol (IN) Pointer to the buffer containing the new symbol.
new_symbol_seqno (IN) New symbol's sequence number in {0.. n-1} range.
store_symbol (IN) true if the function needs to allocate memory, copy the symbol content in it, and call any required callback. This is typically done when this function is called recursively, for newly decoded symbols, or under special circunstances (e.g. perftool).

Returns:

Completion status (LDPC_OK or LDPC_ERROR).

ldpc_error_status LDPCFecSession::DecodingStepWithSymbol (void * symbol_canvas[], void * new_symbol, int new_symbol_seqno)

Perform a new decoding step thanks to the newly received symbol. Same as the other DecodingStepWithSymbol method, without the store_symbol argument (prefered solution).

Parameters:

symbol_canvas (IN-OUT) Global array of received or rebuilt source symbols (parity symbols need not be stored here). This is a table of k pointers to buffers. This array must be cleared (memset(0)) upon the fi rst call to this function. It will be automatically updated, with pointers to symbols received or decoded, by this function.
new_symbol (IN) Pointer to the buffer containing the new symbol.
new_symbol_seqno (IN) New symbol's sequence number in {0.. n-1} range.

Returns:

Completion status (LDPC_OK or LDPC_ERROR).

bool LDPCFecSession::SymbolAlreadyKnown (void * symbol_canvas[], int new_symbol_seqno)

Returns true if the symbol has already been received or decoded (i.e. if it is already known), false otherwise.

Parameters:

symbol_canvas (IN) Array of received/rebuilt source symbols.
new_symbol_seqno (IN) New symbol's sequence number in {0.. n-1} range.

Returns:

TRUE if this symbol has already been received or decoded.

bool LDPCFecSession::IsDecodingComplete (void * symbol_canvas[])
Checks if all DATA symbols have been received/rebuilt.

Parameters:
symbol_canvas (IN) Array of received/rebuilt source symbols.

Returns:
TRUE if all DATA symbols have been received or decoded.

unsigned int LDPCFecSession::GetNbXor (void) `[inline]`
Returns the number of XOR operations performed since last reset. The counter will not distinguish between 64-bit XORs (with 64-bit architectures), 32-bit XORs, and 8-bit XORs.

Returns:
number of 64/32/8-bit XOR operations

void LDPCFecSession::ResetNbXor (void) `[inline]`
Resets the XOR counter.

Author

Generated automatically by Doxygen for ldpc from the source code.