CBUFF-Module

Generated by Doxygen 1.7.1

Mon Feb 6 2012 21:49:10

Contents

Chapter 1

Main Page

1.1 Introduction

The CBUFF Module is designed to be a universal 'unsigned char' circular buffer module.

It is designed to be processor architecture independant, and allows the programmer using it to maintain control of the resources the module needs whilst only requiring minimal resources itself. Specifically defined for microcontroller based applications with minimal flash and RAM memory.

1.2 Contact Information

For more information and the latest release, please visit this projects home page at http://codinghead.github.com/cbuff-module To participate in the project or for other enquiries, please contact Stuart Cording at codinghead@gmail.com

1.3 Licensing Information

Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Author

Stuart Cording aka CODINGHEAD

Note

- 7th Nov 2010 - removed versioning info from file - versioning is now done in GIT

- V0.03 7th May 2010 - renamed all API calls and typedefs so that circular buffer related function names, data types etc. begin with "cbuff".
  - removed Summary of CBUFF_OVERRUN in cbuff.h as it was not being used (possibilty of overrun was removed in V0.01)
  - moved usage of CBUFF_OPEN into the CBUFFOBJ->localFlag instead of using CBUFFOBJ->bufferNumber high bit.
  - Clarified and improved comments for heading of each function
  - Fixed bug in cbuffDestroy which caused "Bus Exception. Unimplemented RAM memory access" on the PIC32 due to pointer to NULL being used for the evaluation of the "bufferNumber" element. Problem should have been apparent on other architectures too.

- V0.02 16th March 2010 - re-wrote the API to allow the creation of buffers to which a handle can then be obtained with the "open" function"

Chapter 2

Todo List

Global cbuffClearBuffer(HCBUFF hCircBuffer) Consider renaming this function to 'cbuffResetBuffer()' and using this function name for the function to acutally clear all of the data in the buffer.

Global cbuffGetByte(HCBUFF hCircBuffer, CBUFF *data) Check if *data doesn't need a const to prevent the function modifying the pointer.

Global cbuffPeekHead(HCBUFF hCircBuffer, CBUFF *data) Check if *data doesn't need a const to prevent the function modifying the pointer.

Global cbuffPeekTail(HCBUFF hCircBuffer, CBUFF *data) Check if *data doesn't need a const to prevent the function modifying the pointer.

Chapter 3

Module Index

3.1 Modules

Here is a list of all modules:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Module Documentation

5.1 Initialise/Deinitialise Functions

Functions

- void cbuffInit (void)

  Initialises the circular buffer module.

- void cbuffDeinit (void)

  Deinitialises the circular buffer module.

5.1.1 Detailed Description

CBUFF Module's functions used to initialise and deinitialise the module.

5.1.2 Function Documentation

5.1.2.1 void cbuffDeinit (void)

Deinitialises the circular buffer module.

Initialises all global variables needed by the CBUFF module. Must be called before using any of the other functions in this module. No check will be made to ensure that you did actually initialise the module before using it, so it is up to you!

See also

cbuffInit()

Note

1. Caller is reposible for returning all current handles and deallocating all buffers before calling cbuffDeinit()

Warning

- The content of any buffers will remain in memory after this function is called. If you have any data there that you don't want other to see, ensure that you clear the buffer's contents before deinitialising this module.

5.1.2.2 void cbuffInit (void)

Initialises the circular buffer module.

Initialises all global variables needed by the CBUFF module. Must be called before using any of the other functions in this module. No check will be made to ensure that you did actually initialise the module before using it, so it is up to you!

See also

cbuffDeinit()

Note

1. Must be called before using any functions in the CBUFF module

5.2 Create/Destroy Functions

Functions

- CBUFFNUM cbuffCreate (CBUFF *buffer, unsigned int sizeOfBuffer, CBUFFOBJ *newCircBufferObj)

  Creates a new buffer object and adds it to the global linked list of buffers.

- unsigned char cbuffDestroy (CBUFFNUM bufferNumber)

  Destroys an exisiting buffer object and removes it to the global linked list of buffers.

5.2.1 Detailed Description

CBUFF Module's functions used to create and destroy buffer objects

5.2.2 Function Documentation

5.2.2.1 CBUFFNUM cbuffCreate (CBUFF * *buffer*, unsigned int *sizeOfBuffer*, CBUFFOBJ * *newCircBufferObj*)

Creates a new buffer object and adds it to the global linked list of buffers.

New buffer object will be added to the global linked list of buffers providing that there is space for another buffer. In total 16 buffers can be supported simultaneously.

See also

cbuffDestroy()

Parameters

buffer - a buffer defined by the caller

sizeOfBuffer - size of aformentioned buffer in bytes

newCircBufferObj - buffer object to insert into linked list of buffer objects

Return values

0 - failed to create the buffer

>=1 - the number asigned to the buffer created

Note

1. The CBUFF module can only handle up to a maximum of 16 buffers
2. It is recommended that the 'sizeOfBuffer' should always be at least 3 or greater to be useful. This will, however, not be checked by this function.

5.2.2.2 unsigned char cbuffDestroy (CBUFFNUM *bufferNumber*)

Destroys an exisiting buffer object and removes it to the global linked list of buffers.

The buffer object will be removed from the global linked list of buffers, providing that the buffer exists and that the buffer has been closed.

See also

cbuffCreate()

Parameters

bufferNumber - number of buffer to destroy as returned by cbuffClose()

Return values

CBUFF_DESTROY_FAIL - failed to destroy requested object

CBUFF_DESTROY_OK - destroyed requested object successfully

Note

1. Caller must have 'created' (cbuffCreate()) at least one circular buffer object before calling this function
2. Destroying a buffer object does not delete the buffer's content, nor free the associated CBUFFOBJ memory used
3. An open buffer cannot be destroyed. Such a case returns a FAIL

5.3 Open/Close Functions

Functions

- HCBUFF cbuffOpen (CBUFFNUM bufferNumber)

  Opens a circular buffer for use by caller and initialises an HCBUFFOBJ handle to it.

- CBUFFNUM cbuffClose (HCBUFF hCircBuffer)

  Closes a circular buffer and releases the handle to it.

5.3.1 Detailed Description

CBUFF Module's functions used to open and close buffer objects that have been created.

5.3.2 Function Documentation

5.3.2.1 CBUFFNUM cbuffClose (HCBUFF *hCircBuffer*)

Closes a circular buffer and releases the handle to it.

The buffer requested will be closed and will no longer be available for use. Attempting to put or get data will fail until the buffer is reopened. Any data that is in the memory where the buffer exists will be retained.

See also

cbuffOpen()

Parameters

hCircBuffer - handle of the open buffer to be closed

Return values

>0 - number of buffer object closed if buffer was open

0 - if the buffer object was not open

Note

1. Caller must have 'allocated' and opened at least one circular buffer object before calling this function

5.3.2.2 HCBUFF cbuffOpen (CBUFFNUM *bufferNumber*)

Opens a circular buffer for use by caller and initialises an HCBUFFOBJ handle to it.

The buffer requested will be initialised for first use. The buffer requested by bufferNumber must exist, otherwise this function will fail.

See also

cbuffClose()

Parameters

bufferNumber - number of an existing buffer to use

Return values

NULL - if buffer couldn't be created

handle - if buffer was created properly

Note

1. Caller must have created (cbuffCreate()) at least one circular buffer object before calling this function

5.4 Put/Get Functions

Functions

- unsigned char cbuffPutByte (HCBUFF hCircBuffer, CBUFF data)
  Puts a single CBUFF into the buffer.
- unsigned char cbuffGetByte (HCBUFF hCircBuffer, CBUFF *data)
  Removes and returns one CBUFF from chosen buffer.
- unsigned int cbuffPutArray (HCBUFF hCircBuffer, const CBUFF *data, unsigned int noOfBytes)
  Allows more than one piece of data to be written into the buffer.
- unsigned int cbuffGetArray (HCBUFF hCircBuffer, CBUFF *data, unsigned int noOfBytes)
  Allows more than one piece of data to be read from the buffer.

5.4.1 Detailed Description

CBUFF Module's functions used to put data into and retrieve date from buffers that have been created and open. Data can be either added/retrieved on a single piece of data basis, or multiple pieces of data basis.

5.4.2 Function Documentation

5.4.2.1 unsigned int cbuffGetArray (HCBUFF *hCircBuffer*, CBUFF * *data*, unsigned int *noOfBytes*)

Allows more than one piece of data to be read from the buffer.

This function allows the caller to read as much data as possible from the chosen buffer object until either all the data has been read, or the destination array is full.

See also

cbuffPutArray(), cbuffPutByte(), cbuffGetByte()

Parameters

hCircBuffer - handle to HCBUFF variable for this buffer instance

data - pointer to location to store data read from buffer

noOfBytes - number of bytes being requested to be read from buffer

Returns

number of bytes actually read from the buffer (may be 0 if buffer is empty)

Note

1. openCircBuffer() must have been successfully called before using this function
2. This function doesn't allow the buffer to underflow
3. User must ensure that CBUFF * data points to a space of free memory large enough to accomodate noOfBytes of data

5.4.2.2 unsigned char cbuffGetByte (HCBUFF *hCircBuffer,* CBUFF * *data*)

Removes and returns one CBUFF from chosen buffer.

A single byte is removed from the buffer indicated by hCircBuffer and written into * data. This function will not allow the buffer to underflow. The buffer must exist and be open before calling this function.

See also

cbuffGetArray(), cbuffPutByte(), cbuffPutArray()

Parameters

hCircBuffer - handle to HCBUFF variable for this buffer instance

data - pointer to variable to store read byte

Return values

CBUFF_GET_OK - operation completed successfully

CBUFF_GET_FAIL - operation failed due to buffer being empty

Note

- openCircBuffer() must have been successfully called before using this function
- This function does not allow a buffer underflow

Todo

Check if *data doesn't need a const to prevent the function modifying the pointer.

5.4.2.3 unsigned int cbuffPutArray (HCBUFF *hCircBuffer,* const CBUFF * *data,* unsigned int *noOfBytes*)

Allows more than one piece of data to be written into the buffer.

This function allows the caller to write as much data as possible into the chosen buffer object until all the data is consumed, or the buffer becomes full.

See also

cbuffGetArray(), cbuffPutByte(), cbuffGetByte()

Parameters

hCircBuffer - handle to HCBUFF variable for this buffer instance

data - pointer to data to be written to buffer

noOfBytes - number of bytes being requested to write into buffer

Returns

number of bytes actually written in the buffer (may be 0 if buffer is full)

Note

1. openCircBuffer() must have been successfully called before using this function
2. This function will not allow the buffer to overflow

5.4.2.4 unsigned char cbuffPutByte (HCBUFF *hCircBuffer,* CBUFF *data*)

Puts a single CBUFF into the buffer.

The CBUFF value provided will be added to the buffer requested. This function will fail only if the buffer is full.

See also

cbuffPutArray(), cbuffGetByte(), cbuffGetArray()

Parameters

hCircBuffer - handle of the open buffer to be used

data - the CBUFF value to be added to the buffer

Return values

CBUFF_PUT_OK - operation was successful

CBUFF_PUT_FAIL - operation failed due to buffer being full

Note

1. cbuffOpen() must have been successfully called before using this function

5.5 Space/Fill Functions

Functions

- unsigned int cbuffGetSpace (HCBUFF hCircBuffer)

  Returns how much more data room the buffer can accept.

- unsigned int cbuffGetFill (HCBUFF hCircBuffer)

  Returns how much data is in the buffer.

5.5.1 Detailed Description

CBUFF Module's functions used to determine how full a buffer is, or how much space remains in the chosen buffer.

5.5.2 Function Documentation

5.5.2.1 unsigned int cbuffGetFill (HCBUFF *hCircBuffer*)

Returns how much data is in the buffer.

Use this function to find out how much space has been used in the buffer or, alternatively, how much data the buffer contains.

See also

cbuffGetSpace()

Parameters

hCircBuffer - handle to HCBUFF variable for this buffer instance

Returns

number of CBUFF bytes of data in the buffer

Note

1. openCircBuffer() must have been successfully called before using this function

5.5.2.2 unsigned int cbuffGetSpace (HCBUFF *hCircBuffer*)

Returns how much more data room the buffer can accept.

Use this function to find out how much space remains in the chosen buffer.

See also

cbuffGetFill()

Parameters

hCircBuffer - handle to HCBUFF variable for this buffer instance

Returns

amount of space remaining in the buffer

Note

1. openCircBuffer() must have been successfully called before using this function

5.6 Clear Buffer Functions

Functions

- void cbuffClearBuffer (HCBUFF hCircBuffer)
  Resets the buffers head and tail pointers and marks buffer as empty.

5.6.1 Detailed Description

CBUFF Module's functions used to reset the chosen buffer back to an empty state.

5.6.2 Function Documentation

5.6.2.1 void cbuffClearBuffer (HCBUFF *hCircBuffer*)

Resets the buffers head and tail pointers and marks buffer as empty.

The head and tail pointers in the requested buffer will be reset to point at the beginning of the buffer and the buffer will be marked as empty. The data currently in the buffer is, however, ∗not∗ deleted.

Parameters

hCircBuffer - handle to HCBUFF variable for this buffer instance

Note

1. openCircBuffer() must have been successfully called before using this function.
2. The associated buffer itself is not emptied. Any data in the buffer will still be in memory after this function is called, although it won't be accessible by this module anymore because the module thinks the buffer is empty.

Todo

Consider renaming this function to 'cbuffResetBuffer()' and using this function name for the function to acutally clear all of the data in the buffer.

5.7 Peek Buffer Head/Tail Functions

Functions

- unsigned char cbuffPeekTail (HCBUFF hCircBuffer, CBUFF *data)

  Returns value of the next CBUFF that would be read from buffer without actually removing it.

- unsigned char cbuffPeekHead (HCBUFF hCircBuffer, CBUFF *data)

  Returns value of the last data item that was stored in the buffer.

5.7.1 Detailed Description

CBUFF Module's functions used to peek into the head or the tail of the chosen buffer. These functions return the last data value written into the buffer or the next data value that would be read out of the buffer if it were to be read.

5.7.2 Function Documentation

5.7.2.1 unsigned char cbuffPeekHead (HCBUFF *hCircBuffer*, CBUFF * *data*)

Returns value of the last data item that was stored in the buffer.

This function allows the caller to see the most recent piece of data that was written into the buffer (using cbuffPutByte() for example). This could be useful when evaluating data received that uses some sort of 'stop' or 'start' byte as part of the protocol and you want advance warning of this as the data is put into the buffer e.g. new NMEA GPS Data always starts with a ';'

See also

cbuffPeekTail()

Parameters

hCircBuffer - handle to HCBUFF variable for this buffer instance

data - pointer to variable to store read byte

Return values

CBUFF_GET_OK - operation completed successfully

CBUFF_GET_FAIL - operation failed due to buffer being empty

Note

1. openCircBuffer() must have been successfully called before using this function

Todo

Check if ∗data doesn't need a const to prevent the function modifying the pointer.

5.7.2.2 unsigned char cbuffPeekTail (HCBUFF *hCircBuffer*, CBUFF ∗ *data*)

Returns value of the next CBUFF that would be read from buffer without actually removing it.

This function allows the caller to see what piece if data would be returned if it were to be removed from the buffer (using cbuffGetByte() for example). This is particularly useful when evaluating data received that uses some sort of 'stop' or 'start' byte as part of the protocol e.g. new NMEA GPS Data always starts with a ';'

See also

cbuffPeekHead()

Parameters

hCircBuffer - handle to HCBUFF variable for this buffer instance

data - pointer to variable to store read byte

Return values

CBUFF_GET_OK - operation completed successfully

CBUFF_GET_FAIL - operation failed due to buffer being empty

Note

1. openCircBuffer() must have been successfully called before using this function
2. This function does not allow a buffer underflow

Todo

Check if ∗data doesn't need a const to prevent the function modifying the pointer.

5.8 Unput/Unget Buffer Functions

Functions

- unsigned char cbuffUnputByte (HCBUFF hCircBuffer)
  Removes the last data item that was stored in the buffer.
- unsigned char cbuffUngetByte (HCBUFF hCircBuffer)
  Returns the last data item that was removed from the buffer.

5.8.1 Detailed Description

CBUFF Module's functions used to return the last data value read back into the buffer, or to remove the last data value written into the buffer from the buffer.

5.8.2 Function Documentation

5.8.2.1 unsigned char cbuffUngetByte (HCBUFF *hCircBuffer*)

Returns the last data item that was removed from the buffer.

This function allows the caller to return the most recent piece of data that was written into the buffer (using cbuffGetByte() for example).

See also

cbuffUnputByte()

Parameters

hCircBuffer - handle to HCBUFF variable for this buffer instance

Return values

0 (zero) - if successful

non-zero - if there was no data to return

Note

1. openCircBuffer() must have been successfully called before using this function
2. this function only shifts the outPointer back; it doesn't write the data that was read back into the circular buffer. It assumes the data that was read out is still in the buffer. If the data has since been overwritten, i.e. buffer is now full, the function will fail in its attempt
3. This function does not allow underflow of the buffer
4. If the buffer was not filled with data, then either random values or the values left over after a 'cbuffClearBuffer' will be 'ungot'. The buffer can still be 'ungot' until the tail pointer gets back to the point where it reaches the head pointer.

5.8.2.2 unsigned char cbuffUnputByte (HCBUFF *hCircBuffer*)

Removes the last data item that was stored in the buffer.

This function allows the caller to remove the most recent piece of data that was written into the buffer (using cbuffPutByte() for example).

See also

cbuffUngetByte()

Parameters

hCircBuffer - handle to HCBUFF variable for this buffer instance

Return values

0 (zero) - if successful

non-zero - if there was no data to remove

Note

1. openCircBuffer() must have been successfully called before using this function
2. This function does not allow underflow of the buffer
3. If the last data byte has since been removed from the buffer, i.e. the buffer is now empty, this function will fail in its attempt
4. The data itself is not removed; only the head pointer to the buffer is moved back one position

5.9 Data Types Needed For CBUFF Use

Typedefs

- typedef unsigned char CBUFF
- typedef unsigned int CBUFFNUM
- typedef struct CBUFFTYPE CBUFFOBJ
- typedef CBUFFOBJ * HCBUFF

5.9.1 Detailed Description

Data types the user will be required to use to make use of the CBUFF Module's functions.

5.9.2 Typedef Documentation

5.9.2.1 CBUFF

Data type for use to create arrays for use as circular buffers with the CBUFF module

5.9.2.2 CBUFFNUM

Used to hold the unique buffer number that the buffer was assigned at creation with cbuffCreate(), to define which buffer should be used when acquiring a handle with cbuffOpen() and which buffer should be destroyed with cbuffDestroy()

5.9.2.3 CBUFFOBJ

The CBUFFOBJ data type is used to create variables that hold all the information needed by the CBUFF module to keep track of the status of the buffer that has been created. One variable is needed per buffer created. Used by cbuffCreate()

5.9.2.4 HCBUFF

The HCBUFF data type is used to create variables to store handles to buffers that have been opened with cbuffOpen(), or to close them with cbuffClose()

Chapter 6

File Documentation

6.1 cbuff.c File Reference

```
#include "cbuff.h"
```

Defines

- #define CBUFF_MODULE__

Functions

- void cbuffInit (void)

  Initialises the circular buffer module.

- void cbuffDeinit (void)

  Deinitialises the circular buffer module.

- CBUFFNUM cbuffCreate (CBUFF *buffer, unsigned int sizeOfBuffer, CBUFFOBJ *newCircBufferObj)

  Creates a new buffer object and adds it to the global linked list of buffers.

- unsigned char cbuffDestroy (CBUFFNUM bufferNumber)

  Destroys an exisiting buffer object and removes it to the global linked list of buffers.

- HCBUFF cbuffOpen (CBUFFNUM bufferNumber)

  Opens a circular buffer for use by caller and initialises an HCBUFFOBJ handle to it.

- CBUFFNUM cbuffClose (HCBUFF hCircBuffer)

  Closes a circular buffer and releases the handle to it.

- unsigned char cbuffPutByte (HCBUFF hCircBuffer, CBUFF data)

  Puts a single CBUFF into the buffer.

- unsigned int cbuffGetSpace (HCBUFF hCircBuffer)

Returns how much more data room the buffer can accept.

- unsigned int cbuffGetFill (HCBUFF hCircBuffer)
  Returns how much data is in the buffer.
- void cbuffClearBuffer (HCBUFF hCircBuffer)
  Resets the buffers head and tail pointers and marks buffer as empty.
- unsigned char cbuffGetByte (HCBUFF hCircBuffer, CBUFF *data)
  Removes and returns one CBUFF from chosen buffer.
- unsigned char cbuffPeekTail (HCBUFF hCircBuffer, CBUFF *data)
  Returns value of the next CBUFF that would be read from buffer without actually removing it.
- unsigned char cbuffPeekHead (HCBUFF hCircBuffer, CBUFF *data)
  Returns value of the last data item that was stored in the buffer.
- unsigned char cbuffUnputByte (HCBUFF hCircBuffer)
  Removes the last data item that was stored in the buffer.
- unsigned char cbuffUngetByte (HCBUFF hCircBuffer)
  Returns the last data item that was removed from the buffer.
- unsigned int cbuffPutArray (HCBUFF hCircBuffer, const CBUFF *data, unsigned int noOfBytes)
  Allows more than one piece of data to be written into the buffer.
- unsigned int cbuffGetArray (HCBUFF hCircBuffer, CBUFF *data, unsigned int noOfBytes)
  Allows more than one piece of data to be read from the buffer.

6.1.1 Detailed Description

6.1.2 Define Documentation

6.1.2.1 #define CBUFF_MODULE__

Defines that the CBUFF module is present. To be used by other software modules to check for the presence of the CBUFF module.

6.2 cbuff.h File Reference

Defines

- #define CBUFF_GET_OK 0x01
- #define CBUFF_GET_FAIL 0x00
- #define CBUFF_PUT_OK 0x01
- #define CBUFF_PUT_FAIL 0x00
- #define CBUFF_DESTROY_FAIL 0x00
- #define CBUFF_DESTROY_OK 0x01

Typedefs

- typedef unsigned char CBUFF
- typedef unsigned int CBUFFNUM
- typedef struct CBUFFTYPE CBUFFOBJ
- typedef CBUFFOBJ * HCBUFF

6.2.1 Detailed Description

Author

Stuart Cording aka CODINGHEAD

Provides a universal 'unsigned char' circular buffer. All contents within this file are 'public' and to be used by end user

Note

See the git versioning notes for version information

6.2.2 Define Documentation

6.2.2.1 #define CBUFF_DESTROY_FAIL 0x00

Signals that cbuffDestroy() failed to deallocate the requested buffer object

6.2.2.2 #define CBUFF_DESTROY_OK 0x01

Signals that cbuffDestroy() successfully deallocated the requested object

6.2.2.3 #define CBUFF_GET_FAIL 0x00

Signals that cbuffGetByte(), cbuffPeekTail(), and cbuffPeekHead() functions failed to read a byte

6.2.2.4 #define CBUFF_GET_OK 0x01

Signals that cbuffGetByte(), cbuffPeekTail(), and cbuffPeekHead() functions successfully read a byte

6.2.2.5 #define CBUFF_PUT_FAIL 0x00

Signals that cbuffPutByte() function failed to write a byte - most likely a sign that the chosen buffer is full

6.2.2.6 #define CBUFF_PUT_OK 0x01

Signals that cbuffPutByte() function successfully wrote a byte

Index