BitStream

Generated by Doxygen 1.8.8

Sun Jan 21 2018 22:20:51

Contents

1 Namespace Index **1**

1.1 Namespace List . . . 1

2 Class Index **1**

2.1 Class List . . . 1

3 File Index **1**

3.1 File List . . . 1

4 Namespace Documentation **2**

4.1 Bits Namespace Reference . . . 2

4.1.1 Detailed Description . . . 3

4.1.2 Typedef Documentation . . . 4

4.1.3 Function Documentation . . . 4

5 Class Documentation **5**

5.1 Bits::Block< NBITS > Class Template Reference . . . 5

5.1.1 Detailed Description . . . 6

5.1.2 Member Typedef Documentation . . . 6

5.1.3 Constructor & Destructor Documentation . . . 6

5.1.4 Member Function Documentation . . . 6

5.1.5 Friends And Related Function Documentation . . . 7

5.1.6 Member Data Documentation . . . 8

5.2 Bits::Float Class Reference . . . 8

5.2.1 Constructor & Destructor Documentation . . . 8

5.2.2 Member Function Documentation . . . 9

5.2.3 Friends And Related Function Documentation . . . 9

5.3 Bits::Stream Class Reference . . . 9

5.3.1 Detailed Description . . . 11

5.3.2 Member Typedef Documentation . . . 11

5.3.3 Constructor & Destructor Documentation . . . 11

5.3.4 Member Function Documentation . . . 12

5.3.5 Friends And Related Function Documentation . . . 13

5.3.6 Member Data Documentation . . . 14

5.4 Bits::varBlock Class Reference . . . 14

5.4.1 Detailed Description . . . 16

5.4.2 Member Typedef Documentation . . . 16

5.4.3 Constructor & Destructor Documentation . . . 16

5.4.4 Member Function Documentation . . . 16

5.4.5 Friends And Related Function Documentation . . . 17

5.4.6 Member Data Documentation 17

6 File Documentation **18**

6.1 BitBase.h File Reference 18

6.2 BitBlock.h File Reference 19

6.3 BitFloat.h File Reference 19

6.4 BitStream.h File Reference 19

6.5 Exemple1.cpp File Reference 19

6.5.1 Function Documentation 20

6.6 Exemple2.cpp File Reference 20

6.6.1 Function Documentation 20

6.7 Exemple3.cpp File Reference 20

6.7.1 Function Documentation 20

Index **21**

1 Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Bits **2**

2 Class Index

2.1 Class List

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

Bits::Block< NBITS > **5**

Bits::Float **8**

Bits::Stream **9**

Bits::varBlock **14**

3 File Index

3.1 File List

Here is a list of all files with brief descriptions:

BitBase.h **18**

BitBlock.h **19**

BitFloat.h 19

BitStream.h 19

Exemple1.cpp 19

Exemple2.cpp 20

Exemple3.cpp 20

4 Namespace Documentation

4.1 Bits Namespace Reference

Classes

- struct **BinaryArray**

  classe technique intermédiaire pour affichage en binaire d'un tableau d'objets bruts dans le flux cette classe est utilisée automatiquement par l'intermédiaire de la fonction Binary(...)
- struct **BinaryObject**

  classe technique intermédiaire pour affichage en binaire d'objets bruts dans le flux cette classe est utilisée automatiquement par l'intermédiaire de la fonction Binary(...)
- class Block
- struct **BlockTypeImpl**
- struct **BlockTypeImpl**< **1** >
- struct **BlockTypeImpl**< **2** >
- struct **BlockTypeImpl**< **3** >
- struct **BlockTypeImpl**< **4** >
- class Float
- class Stream
- class varBlock

Typedefs

- using Size_t = unsigned int
- using Byte = unsigned char
- using Bit = bool

Functions

- template<class T >
  BinaryObject< T > Binary (const T &v, const int pack=0, const int offset=0, const int maxbit=0)

  fonction à utiliser pour affichage en binaire d'un objet dans le flux.
- template<typename T >
  T mask (const Size_t Position, const Size_t Width)

  construction d'un masque de Width bits décalé de Position bits à gauche. Pour Width=0, le masque est sur les bits [0,Width-1] où 0 est le LSB. Si Position + Width dépasse 8∗sizeof(T), le masque est tronqué au-delà. Position doit être strictement intérieur à 8∗sizeof(T). Width doit être inférieur ou égal à 8∗sizeof(T).
- template<typename T >
  T bitmask (const Size_t Position)

  construction du masque associé au bit i (0 = LSB), à savoir tous les bits à 0 sauf le bit i. Position doit être strictement intérieur à 8∗sizeof(T).
- template<typename T >
  Bit get (const T &x, const Size_t Position)

récupère le bit à la position pos (pos = 0 est le LSB) Position doit être strictement intérieur à 8∗sizeof(T).

- template<typename T >
  T set (T x, Size_t Position, Bit Value)

  retourne x dans lequel la valeur du bit à la position Position est fixé à la valeur Value (0=LSB). Position doit être strictement intérieur à 8∗sizeof(T).
- template<typename T >
  T set (const T &x, const Size_t Position, const Size_t Width, const T &y)

  Retourne x en remplaçant ses bits de [Position,Position+Width-1] par les bits [0,Width-1] de y. Position doit être strictement intérieur à 8∗sizeof(T). Width doit être inférieur ou égal à 8∗sizeof(T). Si Position + Width dépasse 8∗sizeof(T), la copie est tronquée au-delà. Ni x ni y ne sont modifiés par cette fonction.
- template<typename T >
  Size_t MSB (T x)

  calcule le bit de poids le plus fort de x (i.e. le bit d'indice le plus grand différent de 0) Exemples: retourne 1 si x=0001, 2 si x=0011, 3 si x=0110, 4 si x=1001, etc ... Si x=0, alors la fonction retourne 0.

4.1.1 Detailed Description

library: bitstream / BitBase.h (opérations binaires de base) author: pascal mignot (université de Reims) version 1.2: mise-à-jour 01/2018

- passage à un fonctionnement en flux avec << et >> pour les lectures/ecritures dans le flux
- Bits::Block<x> : Block avec x bits valides fixés
- Bits::varBlock : Block avec un nombre de bits valides variable
- ajout d'une fonction Binary pour visualiser les données en binaires dans un flux.
- ajout de tests unitaires pour validation

library: bitstream / BitBlock.h (stockage de blocs binaires) author: pascal mignot (université de Reims) version 1.2: mise-à-jour 01/2018

- passage à un fonctionnement en flux avec << et >> pour les lectures/ecritures dans le flux
- Bits::Block<x> : Block avec x bits valides fixés
- Bits::varBlock : Block avec un nombre de bits valides variable
- ajout d'une fonction Binary pour visualiser les données en binaires dans un flux.
- ajout de tests unitaires pour validation

library: bitstream / complement (codage binaire nombres fractionnaires) author: pascal mignot (université de Reims) version 1.2: mise-à-jour 01/2018

- passage à un fonctionnement en flux avec << et >> pour les lectures/ecritures dans le flux
- Bits::Block<x> : Block avec x bits valides fixés
- Bits::varBlock : Block avec un nombre de bits valides variable
- ajout d'une fonction Binary pour visualiser les données en binaires dans un flux.
- ajout de tests unitaires pour validation

library: bitstream / BitStream.h (gestion de flux de bits) author: pascal mignot (université de Reims) version 1.2: mise-à-jour 01/2018

- passage à un fonctionnement en flux avec << et >> pour les lectures/ecritures dans le flux

- Bits::Block<x> : Block avec x bits valides fixés
- Bits::varBlock : Block avec un nombre de bits valides variable
- ajout d'une fonction Binary pour visualiser les données en binaires dans un flux.
- ajout de tests unitaires pour validation

4.1.2 Typedef Documentation

4.1.2.1 using Bits::Bit = typedef bool

4.1.2.2 using Bits::Byte = typedef unsigned char

4.1.2.3 using Bits::Size_t = typedef unsigned int

4.1.3 Function Documentation

4.1.3.1 template<class T > BinaryObject<T> Bits::Binary (const T & *v*, const int *pack* = 0, const int *offset* = 0, const int *maxbit* = 0)

fonction à utiliser pour affichage en binaire d'un objet dans le flux.

Parameters

pack	groupe les bits par paquets de pack (0 = pas de packing)
offset	commence à offset bit depuis le début de l'objet. Les bits avant l'offset ne sont pas affiché (0 = pas d'offset).
maxbit	affiche maxbit au maximum (0 = tous).

4.1.3.2 template<typename T > T Bits::bitmask (const Size_t *Position*)

construction du masque associé au bit i (0 = LSB), à savoir tous les bits à 0 sauf le bit i. Position doit être strictement intérieur à 8∗sizeof(T).

4.1.3.3 template<typename T > Bit Bits::get (const T & *x*, const Size_t *Position*)

récupère le bit à la position pos (pos = 0 est le LSB) Position doit être strictement intérieur à 8∗sizeof(T).

4.1.3.4 template<typename T > T Bits::mask (const Size_t *Position*, const Size_t *Width*)

construction d'un masque de Width bits décalé de Position bits à gauche. Pour Width=0, le masque est sur les bits [0,Width-1] où 0 est le LSB. Si Position + Width dépasse 8∗sizeof(T), le masque est tronqué au-delà. Position doit être strictement intérieur à 8∗sizeof(T). Width doit être inférieur ou égal à 8∗sizeof(T).

4.1.3.5 template<typename T > Size_t Bits::MSB (T *x*)

calcule le bit de poids le plus fort de x (i.e. le bit d'indice le plus grand différent de 0) Exemples: retourne 1 si x=0001, 2 si x=0011, 3 si x=0110, 4 si x=1001, etc ... Si x=0, alors la fonction retourne 0.

4.1.3.6 template<typename T > T Bits::set (T *x*, Size_t *Position*, Bit *Value*)

retourne x dans lequel la valeur du bit à la position Position est fixé à la valeur Value (0=LSB). Position doit être strictement intérieur à 8∗sizeof(T).

4.1.3.7 template<typename T > T Bits::set (const T & *x*, const Size_t *Position*, const Size_t *Width*, const T & *y*)

Retourne x en remplaçant ses bits de [Position,Position+Width-1] par les bits [0,Width-1] de y. Position doit être strictement intérieur à 8∗sizeof(T). Width doit être inférieur ou égal à 8∗sizeof(T). Si Position + Width dépasse 8∗sizeof(T), la copie est tronquée au-delà. Ni x ni y ne sont modifiés par cette fonction.

5 Class Documentation

5.1 Bits::Block< NBITS > Class Template Reference

`#include <BitBlock.h>`

Public Types

- using Type = typename BlockTypeImpl<(NBITS > 0)+(NBITS > 8)+(NBITS > 16)+(NBITS > 32)>::Type

  Type sous-jacent utilisé pour stocker les bits du Bits::Block.

Public Member Functions

Constructeurs

- Block ()

  constructeur par défaut : nb bits valides par défaut, valeur à 0
- Block (Type bits_to_store)

  constructeur par valeur. les bits au-delà du support valide sont perdus.

utilitaires

- Type mask () const

  retourne un mask binaire à 1 sur tous les bits valides et à 0 hors valide.
- void clear ()

  efface tous les bits du bloc (y compris hors valide).
- void clean ()

  met à 0 les bits hors valide.

setters

- void set (Type bits_to_store)

  stocke sans modifier le nombre de bits valides (hors valide non nettoyés)
- void set_bit (Size_t Position, Bit BitValue)

  fixe la valeur du bit Position à BitValue, mais seulement si Position est dans le support valide. Donc, il n'est possible de fixe un bit à l'extérieur des bits valides.

getters

- Type get () const

  retourne le contenu du Block sous forme dans un entier de la taille du support sous-jacent.
- Type get_raw () const

  retourne le contenu brut du Block sous forme dans un entier de la taille du support sous-jacent. Fonction utilisée dans la validation de la classe.
- Size_t get_valid () const

  retourne le nombre de bits valides.

Protected Attributes

- Type bits

opérateurs

- void RotateLeft (Byte r)

  effectue une rotation circulaire à gauche sur le support valide uniquement. exemple: bits = 00001101, NBITS=4 conduit à bits = 00001011
- void RotateRight (Byte r)

  effectue une rotation circulaire à droite sur le support valide uniquement. exemple: bits = 00001101, NBITS=4 conduit à bits = 00001110
- bool operator== (const Block< NBITS > &a, const Block< NBITS > &b)

  compare deux Block sur la seule base de leurs bits valides Ils peuvent donc avoir un nombre de bits valides différents et être égaux s'ils stockent le même nombre binaire.
- Block< NBITS > operator^ (const Block< NBITS > &a, const Block< NBITS > &b)

  retourne le ET logique entre deux Blocks. Leurs nombres de bit valides peuvent être différents
- Block< NBITS > operator+ (const Block< NBITS > &x1, const Block< NBITS > &x2)

  opérateur + entre deux Blocks. L'opération ne fait jamais gagner de précision. S'il y a une retenue sur le résultat de l'opération, celle-ci est perdue. Autrement dit, l'addition se fait modulo 2^max(_valid). Le Block renvoyé a le nombre de bit valide du plus grand des deux.
- Block< NBITS > operator- (const Block< NBITS > &x1, const Block< NBITS > &x2)

  opérateur - entre deux Blocks. L'opération ne fait jamais gagner de précision. Si le résultat est négatif, on obtient le complément à 1. Le Block renvoyé a le nombre de bit valide du plus grand des deux.
- std::ostream & operator<< (std::ostream &os, const Block< NBITS > &x)

  surcharge pour affichage dans un flux de sortie

5.1.1 Detailed Description

template<int NBITS>class Bits::Block< NBITS >

class Bits::Block classe permettant de définir un paquet de bits de 1 à 64 bitsde taille fixe les bits valides doivent sont stockés dans les LSBs.

5.1.2 Member Typedef Documentation

5.1.2.1 template<int NBITS> using Bits::Block< NBITS >::Type = typename BlockTypeImpl<(NBITS > 0) + (NBITS > 8) + (NBITS > 16) + (NBITS > 32)>::Type

Type sous-jacent utilisé pour stocker les bits du Bits::Block.

5.1.3 Constructor & Destructor Documentation

5.1.3.1 template<int NBITS> Bits::Block< NBITS >::Block ( ) [inline]

constructeur par défaut : nb bits valides par défaut, valeur à 0

5.1.3.2 template<int NBITS> Bits::Block< NBITS >::Block (Type *bits_to_store*) [inline]

constructeur par valeur. les bits au-delà du support valide sont perdus.

5.1.4 Member Function Documentation

5.1.4.1 template<int NBITS> void Bits::Block< NBITS >::clean ( ) [inline]

met à 0 les bits hors valide.

5.1.4.2 template<int NBITS> void Bits::Block< NBITS >::clear ( ) [inline]

efface tous les bits du bloc (y compris hors valide).

5.1.4.3 template<int NBITS> Type Bits::Block< NBITS >::get ( ) const [inline]

retourne le contenu du Block sous forme dans un entier de la taille du support sous-jacent.

5.1.4.4 template<int NBITS> Type Bits::Block< NBITS >::get_raw ( ) const [inline]

retourne le contenu brut du Block sous forme dans un entier de la taille du support sous-jacent. Fonction utilisée dans la validation de la classe.

5.1.4.5 template<int NBITS> Size_t Bits::Block< NBITS >::get_valid ( ) const [inline]

retourne le nombre de bits valides.

5.1.4.6 template<int NBITS> Type Bits::Block< NBITS >::mask ( ) const [inline]

retourne un mask binaire à 1 sur tous les bits valides et à 0 hors valide.

5.1.4.7 template<int NBITS> void Bits::Block< NBITS >::RotateLeft (Byte *r*) [inline]

effectue une rotation circulaire à gauche sur le support valide uniquement. exemple: bits = 00001101, NBITS=4 conduit à bits = 00001011

5.1.4.8 template<int NBITS> void Bits::Block< NBITS >::RotateRight (Byte *r*) [inline]

effectue une rotation circulaire à droite sur le support valide uniquement. exemple: bits = 00001101, NBITS=4 conduit à bits = 00001110

5.1.4.9 template<int NBITS> void Bits::Block< NBITS >::set (Type *bits_to_store*) [inline]

stocke sans modifier le nombre de bits valides (hors valide non nettoyés)

5.1.4.10 template<int NBITS> void Bits::Block< NBITS >::set_bit (Size_t *Position,* Bit *BitValue*) [inline]

fixe la valeur du bit Position à BitValue, mais seulement si Position est dans le support valide. Donc, il n'est possible de fixe un bit à l'extérieur des bits valides.

5.1.5 Friends And Related Function Documentation

5.1.5.1 template<int NBITS> Block<NBITS> operator+ (const Block< NBITS > & *x1,* const Block< NBITS > & *x2*) [friend]

opérateur + entre deux Blocks. L'opération ne fait jamais gagner de précision. S'il y a une retenue sur le résultat de l'opération, celle-ci est perdue. Autrement dit, l'addition se fait modulo 2^max(_valid). Le Block renvoyé a le nombre de bit valide du plus grand des deux.

5.1.5.2 template<int NBITS> Block<NBITS> operator- (const Block< NBITS > & *x1,* const Block< NBITS > & *x2*) [friend]

opérateur - entre deux Blocks. L'opération ne fait jamais gagner de précision. Si le résultat est négatif, on obtient le complément à 1. Le Block renvoyé a le nombre de bit valide du plus grand des deux.

5.1.5.3 template<int NBITS> std::ostream& operator<< (std::ostream & *os,* const Block< NBITS > & *x*) [friend]

surcharge pour affichage dans un flux de sortie

5.1.5.4 template<int NBITS> bool operator== (const Block< NBITS > & *a*, const Block< NBITS > & *b*) [friend]

compare deux Block sur la seule base de leurs bits valides Ils peuvent donc avoir un nombre de bits valides différents et être égaux s'ils stockent le même nombre binaire.

5.1.5.5 template<int NBITS> Block<NBITS> operator^ (const Block< NBITS > & *a*, const Block< NBITS > & *b*) [friend]

retourne le ET logique entre deux Blocks. Leurs nombres de bit valides peuvent être différents

5.1.6 Member Data Documentation

5.1.6.1 template<int NBITS> Type Bits::Block< NBITS >::bits [protected]

support sous-jacent de stockage (typiquement uint8_t, uint16_t, uint32_t, uint64_t) déduit de NBITS et utilisé pour stocker les bits du bloc.

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

- BitBlock.h

5.2 Bits::Float Class Reference

`#include <BitFloat.h>`

Public Member Functions

- Float (double v)
- int getValidBits () const
- Float ()

  constructeur par défaut: construit un flottant binaire fractionnaire égal à 0
- void shift (uint32_t nshift=1)
- bool push (Bit v)
- double get () const

  retourne le double associé au binaire fractionnaire complet
- double get (int first, int nb) const

Friends

- std::ostream & operator<< (std::ostream &os, const Float &f)

5.2.1 Constructor & Destructor Documentation

5.2.1.1 Bits::Float::Float (double *v*) [inline]

constructeur à partir d'un double: remplit le Bits:Float à partir d'un double logiquement, le nombre de bits valide à la fin de ce constructeur est 53

5.2.1.2 Bits::Float::Float ( ) [inline]

constructeur par défaut: construit un flottant binaire fractionnaire égal à 0

5.2.2 Member Function Documentation

5.2.2.1 double Bits::Float::get ( ) const [inline]

retourne le double associé au binaire fractionnaire complet

5.2.2.2 double Bits::Float::get (int *first*, int *nb*) const [inline]

retourne le double associé au binaire fractionnaire allant des bits first é first+nb-1 bit de poids le plus fort = 1

5.2.2.3 int Bits::Float::getValidBits ( ) const [inline]

retourne le nombre de bits valide dans le mot Rappel: devrait être en permanence au minimum à qmin (cf poly), ou plus si q est incrémenté. Push pour en ajouter

5.2.2.4 bool Bits::Float::push (Bit *v*) [inline]

ajoute un bit en dernière position du binaire fractionnaire. augmente le nombre de bits valide de 1.

5.2.2.5 void Bits::Float::shift (uint32_t *nshift* = 1) [inline]

shift le nombre binaire à gauche de nshift bit = fait perdre les nshift MSB. dans l'algorithme, associé au décalage de la position de départ. réduit le nombre de bits valide du nombre de shift (par défaut 1)

5.2.3 Friends And Related Function Documentation

5.2.3.1 std::ostream& operator<< (std::ostream & *os*, const Float & *f*) [friend]

surcharge d'un Bits::Float dans le flux de sortie l'affiche sous forme binaire précédé d'un point le nombre de bits affichs correspond au nombre de bits valides

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

- BitFloat.h

5.3 Bits::Stream Class Reference

`#include <BitStream.h>`

Public Member Functions

- Stream (const Size_t base_storage_size=alloc_unit_size)
  constructeur. L'argument est la taille par défait de la zone de stockage
- ~Stream ()
  destructeur
- void status () const
  affiche les positions des pointeurs de lecture et d'écriture
- void clear ()

information sur le flux (pour lecture/écriture de fichiers)

remise à zéro des pointeurs de lecture et d'écriture

- void reset ()
- storage_type * get_data () const
  retour du pointeur sur le buffer de données
- char * get_buffer () const
  retourne un pointeur char vers les données du stream.*

- Size_t get_size () const
  retourne le nombre d'unité de stockage occupé par les données dans le stream
- Size_t get_byte_size () const
  retourne le nombre d'octets occupés par les données dans le stream
- Size_t get_bit_size () const
  retourne le nombre de bits occupés par les données dans le stream
- void set_write_pos (const Size_t nBits)

méthodes utiles pour la lecture des données dans le flux.

déplacement du pointeur de lecture en bit depuis le début du flux L'appel seek() ramène le pointeur de lecture au début du flux. offset est l'offset en bit depuis le début du flux. si offset est plus grand que la taille du flux, le pointeur de lecture est placé en fin de flux.

- void seek (Size_t ibit=0)
- void seek_end (Size_t ebit=0)
  déplacement du pointeur de lecture en bit depuis le fin du flux L'appel seek_end() amène le pointeur de lecture à la fin du flux.
- bool end_of_stream () const
  vrai si le pointeur de lecture a atteint la fin des données écrites
- void get_write_pos (Size_t &iPos, Size_t &iBit) const
  récupère la position du curseur de lecture

gestion de la place mémoire pour le stream

retourne la place mémoire réservée pour le stream (en unité du type sous-jacent de stockage)

- Size_t get_storage_size () const
- Size_t get_storage_byte_size () const
  retourne la place mémoire réservée pour le stream en octets.
- Size_t get_storage_bit_size () const
  retourne la place mémoire réservée pour le stream en bits.
- void request_storage_size (Size_t request_size_in_byte)

Protected Types

- using storage_type = uint32_t
  type sous-jacent de stockage pour le stream

Protected Member Functions

- void realloc (Size_t new_size)
  méthode interne de réallocation

Protected Attributes

- storage_type * buff
  pointeur vers la zone de données
- Size_t wPos
  curseur d'écriture: indice du bloc où se trouve le curseur d'écriture
- Size_t wBit
  curseur d'écriture: indice du bit dans buff[wPos] du curseur d'écriture
- Size_t rPos
  voir wPos.
- Size_t rBit
  curseur de lecture: indice du bit dans buff[wPos] du curseur d'écriture
- Size_t storage_size
  voir rPos.

Static Protected Attributes

- static const Size_t storage_unit_size = 8 ∗ sizeof(storage_type)

  nombre de bits qui peuvent être stockés dans le type sous-jacent
- static const Size_t alloc_unit_size = 1024

  pas de réallocation de la zone de données si elle a besoin d'être agrandie

Friends

- template<class T >
  BinaryArray< T > Binary (const Stream &stream, const int pack=0, const int offset=0, const int maxbit=0)

  fonction à utiliser pour affichage en binaire d'un objet dans le flux.

surcharge des opérateurs pour lecture/écriture dans le stream

attention: les opérateurs >> renvoient toujours le nombre de bits lus.

surcharge opérateur de stream pour les bits. écriture d'un bit

- Stream & operator<< (Stream &stream, const Bit &bit)
- bool operator>> (Stream &stream, Bit &b)
- template<int NBITS>
  Stream & operator<< (Stream &stream, const Block< NBITS > &bitblock)
- template<int NBITS>
  Size_t operator>> (Stream &stream, Block< NBITS > &bitblock)
- Stream & operator<< (Stream &stream, const varBlock &bitblock)
- Size_t operator>> (Stream &stream, varBlock &bitblock)
- template<typename T >
  Stream & operator<< (Stream &stream, const T &data)
- template<typename T >
  Size_t operator>> (Stream &stream, T &data)

5.3.1 Detailed Description

class Bits::Stream classe de gestions d'entrée/sortie de bits

5.3.2 Member Typedef Documentation

5.3.2.1 using Bits::Stream::storage_type = uint32_t [protected]

type sous-jacent de stockage pour le stream

5.3.3 Constructor & Destructor Documentation

5.3.3.1 Bits::Stream::Stream (const Size_t *base_storage_size* = alloc_unit_size) [inline]

constructeur. L'argument est la taille par défait de la zone de stockage

5.3.3.2 Bits::Stream::∼Stream ( ) [inline]

destructeur

5.3.4 Member Function Documentation

5.3.4.1 void Bits::Stream::clear ( ) [inline]

efface les données de la zone de stockage (= les mets à zéro) et réinitialise au début de la zone les pointeurs de lecteur et d'écriture. En résultat, le flux est vide. La mémoire déjà allouée n'est pas modifiée. Utile pour recycler un flux.

5.3.4.2 bool Bits::Stream::end_of_stream ( ) const [inline]

vrai si le pointeur de lecture a atteint la fin des données écrites

5.3.4.3 Size_t Bits::Stream::get_bit_size ( ) const [inline]

retourne le nombre de bits occupés par les données dans le stream

5.3.4.4 char* Bits::Stream::get_buffer ( ) const [inline]

retourne un pointeur char* vers les données du stream.

5.3.4.5 Size_t Bits::Stream::get_byte_size ( ) const [inline]

retourne le nombre d'octets occupés par les données dans le stream

5.3.4.6 storage_type* Bits::Stream::get_data ( ) const [inline]

retour du pointeur sur le buffer de données

5.3.4.7 Size_t Bits::Stream::get_size ( ) const [inline]

retourne le nombre d'unité de stockage occupé par les données dans le stream

5.3.4.8 Size_t Bits::Stream::get_storage_bit_size ( ) const [inline]

retourne la place mémoire réservée pour le stream en bits.

5.3.4.9 Size_t Bits::Stream::get_storage_byte_size ( ) const [inline]

retourne la place mémoire réservée pour le stream en octets.

5.3.4.10 Size_t Bits::Stream::get_storage_size ( ) const [inline]

5.3.4.11 void Bits::Stream::get_write_pos (Size_t & *iPos,* Size_t & *iBit*) const [inline]

récupère la position du curseur de lecture

5.3.4.12 void Bits::Stream::realloc (Size_t *new_size*) [inline], [protected]

méthode interne de réallocation

5.3.4.13 void Bits::Stream::request_storage_size (Size_t *request_size_in_byte*) [inline]

realloue la place mémoire allouée pour le stream s'il n'est pas assez grand pour stocker Request bytes. Donc, le laisse inchangé si la place mémoire allouée est déjé assez grande.

5.3.4.14 void Bits::Stream::reset ( ) [inline]

5.3.4.15 void Bits::Stream::seek (Size_t *ibit* = 0) [inline]

5.3.4.16 void Bits::Stream::seek_end (Size_t *ebit* = 0) [inline]

déplacement du pointeur de lecture en bit depuis le fin du flux L'appel seek_end() amène le pointeur de lecture à la fin du flux.

5.3.4.17 void Bits::Stream::set_write_pos (const Size_t *nBits*) [inline]

fixe la position du curseur de lecture. Utile après avoir rechargé les données dans le flux depuis une source externe. Le paramètre nBits est le nombre de bits écrit dans le flux.

5.3.4.18 void Bits::Stream::status ( ) const [inline]

affiche les positions des pointeurs de lecture et d'écriture

5.3.5 Friends And Related Function Documentation

5.3.5.1 template<class T > BinaryArray<T> Binary (const Stream & *stream*, const int *pack* = 0, const int *offset* = 0, const int *maxbit* = 0) [friend]

fonction à utiliser pour affichage en binaire d'un objet dans le flux.

Parameters

pack	groupe les bits par paquets de pack (0 = pas de packing)
offset	commence à offset bit depuis le début de l'objet. Les bits avant l'offset ne sont pas affiché (0 = pas d'offset).
maxbit	affiche maxbit au maximum (0 = tous).

5.3.5.2 Stream& operator<< (Stream & *stream*, const Bit & *bit*) [friend]

5.3.5.3 template<int NBITS> Stream& operator<< (Stream & *stream*, const Block< NBITS > & *bitblock*) [friend]

surcharge opérateur de stream pour les Bits:Block. écriture d'un BitsBlock

5.3.5.4 Stream& operator<< (Stream & *stream*, const varBlock & *bitblock*) [friend]

surcharge opérateur de stream pour les Bits:Block. écriture d'un BitsBlock

5.3.5.5 template<typename T > Stream& operator<< (Stream & *stream*, const T & *data*) [friend]

surcharge opérateur de stream pour les BITs (uintXX_t) écriture d'un uintXX_t

5.3.5.6 bool operator>> (Stream & *stream*, Bit & *b*) [friend]

surcharge opérateur de stream pour les bits. lecture d'un bit

5.3.5.7 template<int NBITS> Size_t operator>> (Stream & *stream*, Block< NBITS > & *bitblock*) [friend]

surcharge opérateur de stream pour les Bits:Block. lecture d'un BitsBlock

5.3.5.8 Size_t operator>> (Stream & *stream*, varBlock & *bitblock*) [friend]

surcharge opérateur de stream pour les Bits:Block. lecture d'un BitsBlock

5.3.5.9 template<typename T > Size_t operator>> (Stream & *stream*, T & *data*) [friend]

surcharge opérateur de stream pour les BITs (uintXX_t) lecture d'un uintXX_t

5.3.6 Member Data Documentation

5.3.6.1 const Size_t Bits::Stream::alloc_unit_size = 1024 [static], [protected]

pas de réallocation de la zone de données si elle a besoin d'être agrandie

5.3.6.2 storage_type* Bits::Stream::buff [protected]

pointeur vers la zone de données

5.3.6.3 Size_t Bits::Stream::rBit [protected]

curseur de lecture: indice du bit dans buff[wPos] du curseur d'écriture

Se déplace automatiquement du nombre de bits associés au nombre de bits lus dans le flux. Seules les données dans la zone entre le début et la position d'écriture peuvent être lues.

5.3.6.4 Size_t Bits::Stream::rPos [protected]

voir wPos.

curseur de lecture: indice du bloc où se trouve le curseur de lecture

5.3.6.5 Size_t Bits::Stream::storage_size [protected]

voir rPos.

taille de la zone de données réservée

5.3.6.6 const Size_t Bits::Stream::storage_unit_size = 8 * sizeof(storage_type) [static], [protected]

nombre de bits qui peuvent être stockés dans le type sous-jacent

5.3.6.7 Size_t Bits::Stream::wBit [protected]

curseur d'écriture: indice du bit dans buff[wPos] du curseur d'écriture

Se déplace automatiquement du nombre de bits correspondant à chaque écriture faite dans le flux

5.3.6.8 Size_t Bits::Stream::wPos [protected]

curseur d'écriture: indice du bloc où se trouve le curseur d'écriture

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

- BitStream.h

5.4 Bits::varBlock Class Reference

`#include <BitBlock.h>`

Public Types

- using Type = uint64_t

Public Member Functions

Constructeurs

- varBlock ()
  constructeur par défaut : nb bits valides par défaut, valeur à 0

- varBlock (Size_t _valid)
  constructeur taille support (valeur non initialisée).
- varBlock (Size_t _valid, Type bits_to_store)
  constructeur taille support + valeur. les bits au-delà du support valide sont perdus.

utilitaires

- Type mask () const
  retourne un mask binaire à 1 sur tous les bits valides et à 0 hors valide.
- void clear ()
  efface tous les bits du bloc (y compris hors valide).
- void clean ()
  met à 0 les bits hors valide.

setters

- void set (Type bits_to_store)
  stocke sans modifier le nombre de bits valides (hors valide non nettoyés)
- void set_bit (Size_t Position, Bit BitValue)
  fixe la valeur du bit Position à BitValue, mais seulement si Position est dans le support valide. Donc, il n'est possible de fixe un bit à l'extérieur des bits valides.
- void set_valid (Size_t v)
  change le nombre de bits valides. Si le nombre de bits augmente, il est garanti que les bits gagnés sont des 0. Si le nombre de bits diminue, les bits perdus à l'extérieur du support sont mis à zéro.

getters

- Type get () const
  retourne le contenu du Block sous forme dans un entier de la taille du support sous-jacent.
- Type get_raw () const
  retourne le contenu brut du Block sous forme dans un entier de la taille du support sous-jacent. Fonction utilisée dans la validation de la classe.
- Size_t get_valid () const
  retourne le nombre de bits valides

Protected Attributes

- Size_t valid
  nombre de bits valides
- Type bits
  stockage des bits

opérateurs

- void RotateLeft (Size_t r)
  effectue une rotation circulaire à gauche sur le support valide uniquement. exemple: bits = 00001101, NBITS=4 conduit à bits = 00001011
- void RotateRight (Size_t r)
  effectue une rotation circulaire à droite sur le support valide uniquement. exemple: bits = 00001101, NBITS=4 conduit à bits = 00001110
- bool operator== (const varBlock &a, const varBlock &b)
  compare deux Block sur la seule base de leurs bits valides Ils peuvent donc avoir un nombre de bits valides différents et être égaux s'ils stockent le même nombre binaire.
- varBlock operator^ (const varBlock &a, const varBlock &b)
  retourne le ET logique entre deux Blocks. Leurs nombres de bit valides peuvent être différents
- varBlock operator+ (const varBlock &x1, const varBlock &x2)

opérateur + entre deux Blocks. L'opération ne fait jamais gagner de précision. S'il y a une retenue sur le résultat de l'opération, celle-ci est perdue. Autrement dit, l'addition se fait modulo 2^max(_valid). Le Block renvoyé a le nombre de bit valide du plus grand des deux.

- varBlock operator- (const varBlock &x1, const varBlock &x2)

  opérateur - entre deux Blocks. L'opération ne fait jamais gagner de précision. Si le résultat est négatif, on obtient le complément à 1. Le Block renvoyé a le nombre de bit valide du plus grand des deux.

- std::ostream & operator<< (std::ostream &os, const varBlock &x)

  surcharge pour affichage dans un flux de sortie

5.4.1 Detailed Description

class Bits::VarBlock classe permettant de définir un paquet de bits de taille variable (64bits max)

5.4.2 Member Typedef Documentation

5.4.2.1 using Bits::varBlock::Type = uint64_t

5.4.3 Constructor & Destructor Documentation

5.4.3.1 Bits::varBlock::varBlock ( ) [inline]

constructeur par défaut : nb bits valides par défaut, valeur à 0

5.4.3.2 Bits::varBlock::varBlock (Size_t *_valid*) [inline]

constructeur taille support (valeur non initialisée).

5.4.3.3 Bits::varBlock::varBlock (Size_t *_valid,* Type *bits_to_store*) [inline]

constructeur taille support + valeur. les bits au-delà du support valide sont perdus.

5.4.4 Member Function Documentation

5.4.4.1 void Bits::varBlock::clean ( ) [inline]

met à 0 les bits hors valide.

5.4.4.2 void Bits::varBlock::clear ( ) [inline]

efface tous les bits du bloc (y compris hors valide).

5.4.4.3 Type Bits::varBlock::get ( ) const [inline]

retourne le contenu du Block sous forme dans un entier de la taille du support sous-jacent.

5.4.4.4 Type Bits::varBlock::get_raw ( ) const [inline]

retourne le contenu brut du Block sous forme dans un entier de la taille du support sous-jacent. Fonction utilisée dans la validation de la classe.

5.4.4.5 Size_t Bits::varBlock::get_valid ( ) const [inline]

retourne le nombre de bits valides

5.4.4.6 Type Bits::varBlock::mask ( ) const [inline]

retourne un mask binaire à 1 sur tous les bits valides et à 0 hors valide.

5.4.4.7 void Bits::varBlock::RotateLeft (Size_t *r*) [inline]

effectue une rotation circulaire à gauche sur le support valide uniquement. exemple: bits = 00001101, NBITS=4 conduit à bits = 00001011

5.4.4.8 void Bits::varBlock::RotateRight (Size_t *r*) [inline]

effectue une rotation circulaire à droite sur le support valide uniquement. exemple: bits = 00001101, NBITS=4 conduit à bits = 00001110

5.4.4.9 void Bits::varBlock::set (Type *bits_to_store*) [inline]

stocke sans modifier le nombre de bits valides (hors valide non nettoyés)

5.4.4.10 void Bits::varBlock::set_bit (Size_t *Position,* Bit *BitValue*) [inline]

fixe la valeur du bit Position à BitValue, mais seulement si Position est dans le support valide. Donc, il n'est possible de fixe un bit à l'extérieur des bits valides.

5.4.4.11 void Bits::varBlock::set_valid (Size_t *v*) [inline]

change le nombre de bits valides. Si le nombre de bits augmente, il est garanti que les bits gagnés sont des 0. Si le nombre de bits diminue, les bits perdus à l'extérieur du support sont mis à zéro.

5.4.5 Friends And Related Function Documentation

5.4.5.1 varBlock operator+ (const varBlock & *x1,* const varBlock & *x2*) [friend]

opérateur + entre deux Blocks. L'opération ne fait jamais gagner de précision. S'il y a une retenue sur le résultat de l'opération, celle-ci est perdue. Autrement dit, l'addition se fait modulo 2^max(_valid). Le Block renvoyé a le nombre de bit valide du plus grand des deux.

5.4.5.2 varBlock operator- (const varBlock & *x1,* const varBlock & *x2*) [friend]

opérateur - entre deux Blocks. L'opération ne fait jamais gagner de précision. Si le résultat est négatif, on obtient le complément à 1. Le Block renvoyé a le nombre de bit valide du plus grand des deux.

5.4.5.3 std::ostream& operator<< (std::ostream & *os,* const varBlock & *x*) [friend]

surcharge pour affichage dans un flux de sortie

5.4.5.4 bool operator== (const varBlock & *a,* const varBlock & *b*) [friend]

compare deux Block sur la seule base de leurs bits valides Ils peuvent donc avoir un nombre de bits valides différents et être égaux s'ils stockent le même nombre binaire.

5.4.5.5 varBlock operator^ (const varBlock & *a,* const varBlock & *b*) [friend]

retourne le ET logique entre deux Blocks. Leurs nombres de bit valides peuvent être différents

5.4.6 Member Data Documentation

5.4.6.1 Type Bits::varBlock::bits [protected]

stockage des bits

5.4.6.2 Size_t Bits::varBlock::valid [protected]

nombre de bits valides

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

- BitBlock.h

6 File Documentation

6.1 BitBase.h File Reference

```
#include <cstdint>
#include <type_traits>
#include <cassert>
#include <iostream>
```

Namespaces

- Bits

Typedefs

- using Bits::Size_t = unsigned int
- using Bits::Byte = unsigned char
- using Bits::Bit = bool

Functions

- template<class T >
  BinaryObject< T > Bits::Binary (const T &v, const int pack=0, const int offset=0, const int maxbit=0)

  fonction à utiliser pour affichage en binaire d'un objet dans le flux.
- template<typename T >
  T Bits::mask (const Size_t Position, const Size_t Width)

  construction d'un masque de Width bits décalé de Position bits à gauche. Pour Width=0, le masque est sur les bits [0,Width-1] où 0 est le LSB. Si Position + Width dépasse 8∗sizeof(T), le masque est tronqué au-delà. Position doit être strictement intérieur à 8∗sizeof(T). Width doit être inférieur ou égal à 8∗sizeof(T).
- template<typename T >
  T Bits::bitmask (const Size_t Position)

  construction du masque associé au bit i (0 = LSB), à savoir tous les bits à 0 sauf le bit i. Position doit être strictement intérieur à 8∗sizeof(T).
- template<typename T >
  Bit Bits::get (const T &x, const Size_t Position)

  récupère le bit à la position pos (pos = 0 est le LSB) Position doit être strictement intérieur à 8∗sizeof(T).
- template<typename T >
  T Bits::set (T x, Size_t Position, Bit Value)

  retourne x dans lequel la valeur du bit à la position Position est fixé à la valeur Value (0=LSB). Position doit être strictement intérieur à 8∗sizeof(T).
- template<typename T >
  T Bits::set (const T &x, const Size_t Position, const Size_t Width, const T &y)

  Retourne x en remplaçant ses bits de [Position,Position+Width-1] par les bits [0,Width-1] de y. Position doit être strictement intérieur à 8∗sizeof(T). Width doit être inférieur ou égal à 8∗sizeof(T). Si Position + Width dépasse 8∗sizeof(T), la copie est tronquée au-delà. Ni x ni y ne sont modifiés par cette fonction.
- template<typename T >
  Size_t Bits::MSB (T x)

  calcule le bit de poids le plus fort de x (i.e. le bit d'indice le plus grand différent de 0) Exemples: retourne 1 si x=0001, 2 si x=0011, 3 si x=0110, 4 si x=1001, etc ... Si x=0, alors la fonction retourne 0.

6.2 BitBlock.h File Reference

```
#include "BitBase.h"
```

Classes

- class Bits::Block< NBITS >
- class Bits::varBlock

Namespaces

- Bits

6.3 BitFloat.h File Reference

```
#include <cstdint>
#include <iostream>
#include "BitBase.h"
```

Classes

- class Bits::Float

Namespaces

- Bits

6.4 BitStream.h File Reference

```
#include <cstring>
#include "BitBase.h"
#include "BitBlock.h"
```

Classes

- class Bits::Stream

Namespaces

- Bits

6.5 Exemple1.cpp File Reference

```
#include <iostream>
#include <random>
#include <vector>
#include "BitStream.h"
```

Functions

- int main (int argc, char ∗argv[])

6.5.1 Function Documentation

6.5.1.1 int main (int *argc,* char ∗ *argv[]*)

6.6 Exemple2.cpp File Reference

```
#include <iostream>
#include <fstream>
#include <random>
#include <vector>
#include "BitStream.h"
```

Functions

- int main (int argc, char ∗argv[])

6.6.1 Function Documentation

6.6.1.1 int main (int *argc,* char ∗ *argv[]*)

6.7 Exemple3.cpp File Reference

```
#include <iostream>
#include <iomanip>
#include <limits>
#include "BitFloat.h"
```

Functions

- int main (int argc, char ∗argv[])

6.7.1 Function Documentation

6.7.1.1 int main (int *argc,* char ∗ *argv[]*)

Index

Binary
    Bits, 4
Bit
    Bits, 4
bitmask
    Bits, 4
Bits, 2
    Binary, 4
    Bit, 4
    bitmask, 4
    Byte, 4
    get, 4
    mask, 4
    set, 4
Byte
    Bits, 4

get
    Bits, 4

mask
    Bits, 4

set
    Bits, 4