Table of Contents

[Classical Board](#)

[Structure](#)

[Array](#)

[Denser Board](#)

[See also](#)

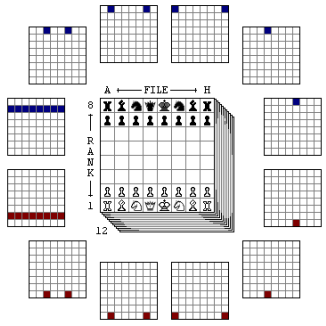
[Forum Posts](#)

[External Links](#)

[References](#)

[What links here?](#)

To represent the board we typically need one bitboard for each [piece-type](#) and [color](#) - likely encapsulated inside a class or structure, or as an [array](#) of bitboards as part of a [position](#) object. A one-bit inside a bitboard implies the existence of a piece of this piece-type on a certain [square](#) - one to one associated by the bit-position [1]:



To be aware of their scalar 64-bit origin, we use so far a type defined unsigned integer U64 in our [C](#)/[C++](#) source snippets, the scalar 64-bit long in [Java](#). Feel free to define a distinct type or wrap U64 into classes for better abstraction and type-safety during compile time.

Classical Board

Those bitboards may part of a central position object which is incrementally updated while making or unmaking moves.

Structure

```
class CBoard
{
   U64 whitePawns;
   U64 whiteKnights;
   U64 whiteBishops;
   U64 whiteRooks;
   U64 whiteQueens;
   U64 whiteKing;

   U64 blackPawns;
   U64 blackKnights;
   U64 blackBishops;
   U64 blackRooks;
   U64 blackQueens;
   U64 blackKing;
   ...
};
```

Array

For better generalization and to avoid branches, one may encapsulate arrays of bitboards. For instance, inside the Beowulf sources (sample from moves.c) one finds a lot of branches on side to move to either fetch white or black piece bitboards, as already criticized by Vincent Diepeveen in 2001 [2] ...

```
  switch (side) {
   case WHITE: tsq = B->whiteRooks; break;
   case BLACK: tsq = B->blackRooks; break;
  }
```

.. where an indexed access with appropriate defined {0,1} color range for the side to move would avoid those branches, per piece-kind, ...

```
  tsq = B->rooks[side];
```

... or over all piece-kinds, ...

```
  tsq = B->pieceBB[nWhiteRook + side];
```

... for instance, on x86 or x86-64, utilizing its addressing modes with base- and scalable index register, plus displacement:

```
   ; rsi pointer to structure, rcx side (0 == White, 1 == Black)
   mov  rax, qword ptr [rsi + rookOffset + 8*rcx]
```

Likely one also keeps some often used redundant union sets like white and black pieces, occupancy or their complement, the empty squares.

```
class CBoard
{
   U64 pieceBB[14];
   U64 emptyBB;
   U64 occupiedBB;
   ...
public:
   enum enumPiece
   {
      nWhite,     // any white piece
      nBlack,     // any black piece
      nWhitePawn, // white Pawn
      nBlackPawn, // white Pawn
      ...
   };

   U64 getPieceSet(PieceType pt) const {return pieceBB[pt];}
   U64 getWhitePawns() const {return pieceBB[nWhitePawn];}
   ...
   U64 getPawns(ColorType ct) const {return pieceBB[nWhitePawn + ct];}
   ...
};
```

On initialization and update of the bitboards, see general setwise operations.

Denser Board

A common alternative to reduce the size of the board structure is to keep two color bitboards and six color independent piece bitboards, which are the union of black and white respective pieces, i.e. all queens. This

space saving requires a cheap intersection of a color and a piece bitboard to get the required pieces of that color only.

```
class CBoard
{
   U64 pieceBB[8];
public:
   enum enumPiece
   {
      nWhite,     // any white piece
      nBlack,     // any black piece
      nPawn,
      nKnight,
      nBishop,
      nRook,
      nQueen,
      nKing
   };

   U64 getPieceSet(PieceType pt) const {return pieceBB[
pieceCode(pt)] & pieceBB[colorCode(pt)];}
   U64 getWhitePawns() const {return pieceBB[nPawn] & pieceBB[
nWhite];}
   ...
   U64 getPawns(ColorType ct) const {return pieceBB[nPawn] &
 pieceBB[ct];}
   ...
};
```

See also

- Color Flipping
- Quad-Bitboards

Forum Posts

- Bit Board Bonkers?? by Dave, rec.games.chess.computer, July 28, 1997
- Bitboard board representation by Eric Oldre, CCC, January 13, 2005
- BitBoard representations of the board by Uri Blass, CCC, October 14, 2007
- Decision concerning board representation by Piotr Lopusiewicz, CCC, May 05, 2013

External Links

- Glenn Gould - Paul Hindemith, Piano Sonata No. 3 - Fugue, YouTube Video

References

1. ^ Bitwise Optimization in Java: Bitfields, Bitboards, and Beyond by Glen Pepicelli, 2005, O'Reilly's OnJava.com
2. ^ On Beowulf - long post by Vincent Diepeveen, CCC, April 04, 2001

What links here?

Page	Date Edited
AlphaZero	Feb 10, 2018
Array	Dec 1, 2016
Bitboard Board-Definition	Jun 23, 2014
Bitboards	Nov 14, 2017
Checks and Pinned Pieces (Bitboards)	Aug 14, 2013
Chess 0.5	Nov 20, 2016
Color Flipping	May 17, 2017
Counter	Nov 13, 2017
Demolito	Mar 1, 2018
Efficient Generation of Sliding Piece Attacks	Nov 5, 2016
General Setwise Operations	Feb 25, 2018
Incremental Updates	Sep 6, 2017
Initial Position	Apr 14, 2018
King Pattern	Nov 15, 2013
Knight Pattern	Feb 23, 2015
Lachex	Jan 7, 2016
Make Move	Mar 2, 2016
Material Tables	May 5, 2017
Occupancy	Sep 19, 2016
Pieces	Feb 19, 2018
Quad-Bitboards	Jan 30, 2017
Rodent	Jan 11, 2018
Rotated Bitboards	Mar 7, 2017
RuyDos	Feb 17, 2018
SEE - The Swap Algorithm	Jun 5, 2017
Simona Tancig	Nov 7, 2012
Sliding Piece Attacks	May 27, 2016
Spector	Nov 11, 2016
Square Attacked By	Jan 20, 2018
Sungorus	Apr 11, 2014
Teki	Mar 29, 2018
Tunguska	Sep 16, 2017
WyldChess	Mar 10, 2018

Page	Date Edited
X-ray Attacks (Bitboards)	Mar 31, 2015

Up one Level