This PDF represents the changes performed while forking mathematical libraries from \*\*uniswap\*\* library to make it compatible with \*\*Fountain\*\* project's `Oracle` contract uniswap npm paths

- "@uniswap\lib\contracts\libraries\FullMath.sol"
- "@uniswap\lib\contracts\libraries\BitMath.sol"
- "@uniswap\lib\contracts\libraries\FixedPoint.sol"

Project's github repo

https://github.com/puls369ar/fountain-solicy-overview/Oracle

# **Changes Description**

In Solidity, type(uint224).max returns the **maximum value** that a uint224 (an unsigned 224-bit integer) can hold. The same we've got in older version inside `FullMath`, `BitMath` and `FixedPoint` libraries, by using uint224(-1)

Also arithmetic operations changed with binary ones to prevent uint256 issues When calculating `pow2`

below find code's \*diff\* between forked and original versions



### Comparing sensitive data, confidential files or internal emails?

Most legal and privacy policies prohibit uploading sensitive data online. Diffchecker Desktop ensures your confidential information never leaves your computer. Work offline and compare documents securely.

**Download Diffchecker Desktop** 

Learn more

# Untitled diff

8 removals

86 lines

8 additions

86 lines

```
1 // SPDX-License-Identifier: GPL-3.0-or-
                                                1 // SPDX-License-Identifier: GPL-3.0-or-
   later
                                                   later
 2 pragma solidity >=0.5.0;
                                                2 pragma solidity >=0.8.20;
                                                3
 3
 4 library BitMath {
                                                4 library BitMath {
       // Returns the 0 indexed position of
                                                      // returns the 0 indexed position of
   the most significant bit of the input x
                                                   the most significant bit of the input x
       // s.t. x >= 2**msb and x < 2**
                                                       // s.t. x >= 2**msb and x < 2**
   (msb+1)
                                                   (msb+1)
       function mostSignificantBit(uint256
                                                      function mostSignificantBit(uint256
 7
   x) internal pure returns (uint8 r) {
                                                   x) internal pure returns (uint8 r) {
           require(x > 0,
                                                          require(x > 0,
 8
                                                8
   'BitMath::mostSignificantBit: zero');
                                                   'BitMath::mostSignificantBit: zero');
                                                9
10
           if (x >=
                                               10
                                                          if (x >=
   11
              x >>= 128;
                                               11
                                                              x >>= 128;
12
               r += 128;
                                               12
                                                              r += 128;
13
           }
                                               13
           if (x >= 0x1000000000000000) {
                                                          if (x >= 0x1000000000000000) {
14
                                               14
15
              x >>= 64;
                                                              x >>= 64;
                                               15
               r += 64;
                                                              r += 64;
16
                                               16
17
           }
                                               17
                                                          }
           if (x >= 0x100000000) {
                                                          if (x >= 0x100000000) {
18
                                               18
19
               x >>= 32;
                                               19
                                                              x >>= 32;
20
               r += 32;
                                               20
                                                              r += 32;
21
                                               21
                                                          if (x >= 0x10000) {
22
           if (x >= 0x10000) {
                                               22
23
              x >>= 16;
                                               23
                                                              x >>= 16;
```

```
24
                                                   24
                                                                   r += 16;
                r += 16;
25
            }
                                                   25
                                                               }
            if (x >= 0x100) {
                                                               if (x >= 0x100) {
26
                                                   26
                x >>= 8;
27
                                                   27
                                                                   x >>= 8;
                                                                   r += 8;
28
                r += 8;
                                                   28
29
                                                   29
            }
                                                               }
30
            if (x >= 0x10) {
                                                   30
                                                               if (x >= 0x10) {
                x >>= 4;
                                                                   x >>= 4;
31
                                                   31
32
                r += 4;
                                                   32
                                                                   r += 4;
33
                                                   33
            }
                                                               if (x >= 0x4) {
34
            if (x >= 0x4) {
                                                   34
35
                x >>= 2;
                                                                   x >>= 2;
                                                   35
                r += 2;
                                                                   r += 2;
36
                                                   36
37
                                                   37
38
           if (x >= 0x2) r += 1;
                                                   38
                                                               if (x >= 0x2) r += 1;
39
       }
                                                   39
                                                           }
                                                   40
40
       // Returns the 0 indexed position of
                                                           // returns the 0 indexed position of
41
                                                   41
   the least significant bit of the input x
                                                       the least significant bit of the input x
42
       // s.t. (x & 2**1sb) != 0 and (x &
                                                           // s.t. (x & 2**1sb) != 0 and (x &
   (2**(1sb) - 1)) == 0)
                                                       (2**(lsb) - 1)) == 0)
       // i.e. the bit at the index is set
                                                   43
                                                           // i.e. the bit at the index is set
43
   and the mask of all lower bits is 0
                                                       and the mask of all lower bits is 0
44
       function leastSignificantBit(uint256
                                                   44
                                                           function leastSignificantBit(uint256
   x) internal pure returns (uint8 r) {
                                                       x) internal pure returns (uint8 r) {
            require(x > 0,
                                                   45
                                                               require(x > 0,
45
    'BitMath::leastSignificantBit: zero');
                                                       'BitMath::leastSignificantBit: zero');
                                                   46
46
47
            r = 255;
                                                   47
                                                               r = 255;
            if (x & type(uint128).max > 0) {
                                                               if (x \& uint128(-1) > 0) {
48
                                                   48
49
                r -= 128;
                                                   49
                                                                   r -= 128;
                                                               } else {
50
            } else {
                                                   50
51
                x >>= 128;
                                                   51
                                                                   x >>= 128;
52
            }
                                                   52
                                                               }
                                                               if (x \& uint64(-1) > 0) {
            if (x & type(uint64).max > 0) {
53
                                                   53
54
                r -= 64:
                                                   54
                                                                   r -= 64;
55
            } else {
                                                               } else {
                                                   55
                x >>= 64;
                                                   56
                                                                   x >>= 64;
56
57
            }
                                                   57
58
            if (x & type(uint32).max > 0) {
                                                   58
                                                               if (x \& uint32(-1) > 0) {
59
                r -= 32:
                                                   59
                                                                   r -= 32;
            } else {
                                                               } else {
60
                                                   60
                x >>= 32;
                                                   61
                                                                   x >>= 32;
61
                                                   62
62
            if (x & type(uint16).max > 0) {
                                                   63
                                                               if (x \& uint16(-1) > 0) {
```

Untitled diff - Diffchecker

10/18/24, 3:42 PM

```
10/18/24, 3:42 PM
                                                         Untitled diff - Diffchecker
    64
                    r -= 16;
                                                                         r -= 16;
                                                         64
                                                                     } else {
    65
                } else {
                                                         65
                     x >>= 16;
    66
                                                         66
                                                                         x >>= 16;
    67
                                                         67
                }
                                                                     }
    68
                if (x & type(uint8).max > 0) {
                                                         68
                                                                     if (x \& uint8(-1) > 0) {
                     r -= 8;
                                                                         r -= 8;
    69
                                                         69
    70
                                                         70
                } else {
                                                                     } else {
    71
                     x >>= 8;
                                                         71
                                                                         x >>= 8;
    72
                                                         72
                if (x \& 0xf > 0) {
                                                                     if (x \& 0xf > 0) {
    73
                                                         73
    74
                     r -= 4;
                                                         74
                                                                         r -= 4;
    75
                } else {
                                                         75
                                                                     } else {
    76
                     x >>= 4;
                                                         76
                                                                         x >>= 4;
    77
                                                         77
                }
                if (x \& 0x3 > 0) {
                                                                     if (x \& 0x3 > 0) {
    78
                                                         78
    79
                     r -= 2;
                                                         79
                                                                         r -= 2;
    80
                } else {
                                                         80
                                                                     } else {
                     x >>= 2;
    81
                                                         81
                                                                         x >>= 2;
    82
                }
                                                         82
                if (x \& 0x1 > 0) r -= 1;
                                                                     if (x \& 0x1 > 0) r -= 1;
    83
                                                         83
    84
            }
                                                         84
                                                                 }
    85 }
                                                         85 }
                                                         86
```



### Comparing sensitive data, confidential files or internal emails?

Most legal and privacy policies prohibit uploading sensitive data online. Diffchecker Desktop ensures your confidential information never leaves your computer. Work offline and compare documents securely.

**Download Diffchecker Desktop** 

Learn more

# Untitled diff



147 lines



147 lines

```
1 // SPDX-License-Identifier: GPL-3.0-or-
   later
 2 pragma solidity >=0.8.20;
 4 import
   '@uniswap/lib/contracts/libraries/FullM
   ath.sol';
 5 import
   '@uniswap/lib/contracts/libraries/Babyl
   onian.sol';
 6 import './BitMath.sol';
 7
 8 // A library for handling binary fixed-
   point numbers
   (https://en.wikipedia.org/wiki/Q_(numbe
   r_format))
 9 library FixedPoint {
       // Range: [0, 2**112 - 1]
10
       // Resolution: 1 / 2**112
11
12
       struct uq112x112 {
13
           uint224 _x;
14
       }
15
16
       // Range: [0, 2**144 - 1]
       // Resolution: 1 / 2**112
17
       struct uq144x112 {
18
19
           uint256 _x;
20
       }
```

```
1 // SPDX-License-Identifier: GPL-3.0-or-
   later
 2 pragma solidity >=0.8.20;
 4 import
   '@uniswap/lib/contracts/libraries/FullM
   ath.sol';
 5 import
   '@uniswap/lib/contracts/libraries/Babyl
   onian.sol';
 6 import './BitMath.sol';
 8 // a library for handling binary fixed
   point numbers
   (https://en.wikipedia.org/wiki/Q_(numbe
   r_format))
 9 library FixedPoint {
10
       // range: [0, 2**112 - 1]
11
       // resolution: 1 / 2**112
12
       struct uq112x112 {
13
           uint224 _x;
14
       }
15
16
       // range: [0, 2**144 - 1]
       // resolution: 1 / 2**112
17
18
       struct uq144x112 {
19
           uint256 _x;
20
       }
```

Untitled diff - Diffchecker

10/18/24, 3:47 PM

```
// Multiply a UQ112x112 by a uint,
                                                          // multiply a UQ112x112 by a uint,
47
                                                  47
   returning a UQ144x112
                                                      returning a UQ144x112
       // Reverts on overflow
                                                          // reverts on overflow
49
       function mul(uq112x112 memory self,
                                                  49
                                                          function mul(uq112x112 memory self,
   uint256 y) internal pure returns
                                                      uint256 y) internal pure returns
   (uq144x112 memory) {
                                                      (uq144x112 memory) {
50
           uint256 z = 0;
                                                  50
                                                              uint256 z = 0;
           require(y == 0 \mid | (z = self._x)
                                                              require(y == 0 \mid | (z = self._x)
51
                                                  51
   * y) / y == self._x, 'FixedPoint::mul:
                                                      * y) / y == self._x, 'FixedPoint::mul:
   overflow');
                                                      overflow');
52
           return uq144x112(z);
                                                  52
                                                              return uq144x112(z);
53
       }
                                                  53
                                                          }
54
                                                  54
       // Multiply a UQ112x112 by an int
                                                          // multiply a UQ112x112 by an int
                                                      and decode, returning an int
   and decode, returning an int
     // Reverts on overflow
                                                        // reverts on overflow
56
                                                  56
57
       function muli(uq112x112 memory
                                                  57
                                                          function muli(uq112x112 memory
   self, int256 y) internal pure returns
                                                      self, int256 y) internal pure returns
   (int256) {
                                                      (int256) {
58
           uint256 z =
                                                  58
                                                              uint256 z =
   FullMath.mulDiv(self._x, uint256(y < 0</pre>
                                                      FullMath.mulDiv(self._x, uint256(y < 0</pre>
   ? -y : y), Q112);
                                                      ? -y : y), Q112);
           require(z < 2**255,
                                                  59
                                                              require(z < 2**255,
59
   'FixedPoint::muli: overflow');
                                                      'FixedPoint::muli: overflow');
           return y < 0? -int256(z):
                                                              return y < 0? -int256(z):
   int256(z);
                                                      int256(z);
                                                  61
                                                          }
61
       }
                                                  62
62
       // Multiply a UQ112x112 by a
                                                          // multiply a UQ112x112 by a
63
                                                  63
   UQ112x112, returning a UQ112x112
                                                      UQ112x112, returning a UQ112x112
      // Lossy
                                                        // lossy
       function muluq(uq112x112 memory
                                                          function muluq(uq112x112 memory
   self, uq112x112 memory other) internal
                                                      self, uq112x112 memory other) internal
   pure returns (uq112x112 memory) {
                                                      pure returns (uq112x112 memory) {
           if (self._x == 0 || other._x ==
                                                              if (self._x == 0 \mid \mid other._x ==
66
                                                  66
   0) {
                                                      0) {
               return uq112x112(0);
                                                                  return uq112x112(0);
67
                                                  67
68
           }
                                                  68
                                                              }
           uint112 upper self =
                                                              uint112 upper self =
                                                  69
   uint112(self._x >> RESOLUTION); // *
                                                      uint112(self._x >> RESOLUTION); // *
           uint112 lower self =
                                                  70
                                                              uint112 lower self =
   uint112(self._x & LOWER_MASK); // *
                                                      uint112(self._x & LOWER_MASK); // *
   2^-112
                                                      2^-112
```

92

// divide a UQ112x112 by a

UQ112x112, returning a UQ112x112

92 // Divide a UQ112x112 by a

UQ112x112, returning a UQ112x112

91

```
93
        function divug(uq112x112 memory
                                                           function divuq(uq112x112 memory
    self, uq112x112 memory other) internal
                                                       self, uq112x112 memory other) internal
    pure returns (uq112x112 memory) {
                                                       pure returns (uq112x112 memory) {
            require(other._x > 0,
                                                    94
                                                                require(other._x > 0,
 94
     'FixedPoint::divug: division by zero');
                                                        'FixedPoint::divuq: division by zero');
                                                                if (self._x == other._x) {
            if (self. x == other. x) {
 95
                                                    95
                 return
                                                    96
                                                                    return
 96
    uq112x112(uint224(Q112));
                                                       uq112x112(uint224(Q112));
                                                    97
 97
            }
                                                                }
                                                                if (self._x <= uint144(-1)) {
 98
            if (self._x <=
                                                    98
    type(uint144).max) { // Changed here
                 uint256 value =
 99
                                                    99
                                                                    uint256 value =
    (uint256(self._x) << RESOLUTION) /</pre>
                                                       (uint256(self._x) << RESOLUTION) /</pre>
    other._x;
                                                       other. x;
                 require(value <=
100
                                                   100
                                                                    require(value <=
    type(uint224).max, 'FixedPoint::divuq:
                                                       uint224(-1), 'FixedPoint::divuq:
    overflow');
                                                       overflow');
101
                 return
                                                   101
                                                                    return
    uq112x112(uint224(value));
                                                       uq112x112(uint224(value));
102
            }
                                                   102
                                                                }
103
                                                   103
104
            uint256 result =
                                                   104
                                                                uint256 result =
    FullMath.mulDiv(Q112, self._x,
                                                       FullMath.mulDiv(Q112, self._x,
    other._x);
                                                       other._x);
                                                                require(result <= uint224(-1),
105
             require(result <=
                                                   105
    type(uint224).max, 'FixedPoint::divuq:
                                                        'FixedPoint::divuq: overflow');
    overflow');
106
            return
                                                   106
                                                                return
    uq112x112(uint224(result));
                                                       uq112x112(uint224(result));
107
        }
                                                   107
                                                           }
108
                                                   108
109
        // Returns a UQ112x112 which
                                                   109
                                                           // returns a UQ112x112 which
    represents the ratio of the numerator
                                                       represents the ratio of the numerator
    to the denominator
                                                       to the denominator
110
        // Can be lossy
                                                   110
                                                           // can be lossy
111
        function fraction(uint256
                                                   111
                                                           function fraction(uint256
    numerator, uint256 denominator)
                                                       numerator, uint256 denominator)
    internal pure returns (uq112x112
                                                       internal pure returns (uq112x112
    memory) {
                                                       memory) {
            require(denominator > 0,
                                                                require(denominator > 0,
112
                                                   112
    'FixedPoint::fraction: division by
                                                        'FixedPoint::fraction: division by
    zero');
                                                       zero');
            if (numerator == 0) return
113
                                                   113
                                                                if (numerator == 0) return
    FixedPoint.uq112x112(0);
                                                       FixedPoint.uq112x112(0);
114
                                                   114
```

```
if (numerator <=</pre>
                                                               if (numerator <= uint144(-1)) {</pre>
115
                                                  115
    type(uint144).max) { // Changed here
                 uint256 result = (numerator
                                                                   uint256 result = (numerator
116
                                                  116
    << RESOLUTION) / denominator;
                                                       << RESOLUTION) / denominator;
117
                 require(result <=
                                                  117
                                                                   require(result <=
    type(uint224).max,
                                                       uint224(-1), 'FixedPoint::fraction:
     'FixedPoint::fraction: overflow');
                                                       overflow');
                return
                                                  118
                                                                   return
118
                                                       uq112x112(uint224(result));
    uq112x112(uint224(result));
119
                                                  119
            } else {
                                                               } else {
                uint256 result =
                                                                   uint256 result =
120
                                                  120
                                                       FullMath.mulDiv(numerator, Q112,
    FullMath.mulDiv(numerator, Q112,
    denominator);
                                                       denominator);
121
                 require(result <=
                                                  121
                                                                   require(result <=
    type(uint224).max,
                                                       uint224(-1), 'FixedPoint::fraction:
    'FixedPoint::fraction: overflow');
                                                       overflow');
                                                  122
122
                return
                                                                   return
    uq112x112(uint224(result));
                                                       uq112x112(uint224(result));
123
            }
                                                  123
                                                               }
124
        }
                                                  124
                                                           }
125
                                                  125
126
        // Take the reciprocal of a
                                                  126
                                                           // take the reciprocal of a
    UQ112x112
                                                       UQ112x112
                                                           // reverts on overflow
127
        // Reverts on overflow
                                                  127
        // Lossy
                                                           // lossy
128
                                                  128
                                                           function reciprocal(uq112x112
129
        function reciprocal(ug112x112
                                                  129
    memory self) internal pure returns
                                                       memory self) internal pure returns
    (uq112x112 memory) {
                                                       (uq112x112 memory) {
            require(self._x != 0,
130
                                                  130
                                                               require(self._x != 0,
     'FixedPoint::reciprocal: reciprocal of
                                                       'FixedPoint::reciprocal: reciprocal of
    zero');
                                                       zero');
131
            require(self._x != 1,
                                                  131
                                                               require(self._x != 1,
     'FixedPoint::reciprocal: overflow');
                                                       'FixedPoint::reciprocal: overflow');
            return uq112x112(uint224(Q224 /
132
                                                  132
                                                               return uq112x112(uint224(Q224 /
    self._x));
                                                       self._x));
133
        }
                                                  133
                                                           }
134
                                                  134
        // Square root of a UQ112x112
135
                                                  135
                                                           // square root of a UQ112x112
        // Lossy between 0/1 and 40 bits
                                                           // lossy between 0/1 and 40 bits
136
                                                  136
137
        function sqrt(uq112x112 memory
                                                  137
                                                           function sqrt(uq112x112 memory
    self) internal pure returns (uq112x112
                                                       self) internal pure returns (uq112x112
    memory) {
                                                       memory) {
            if (self._x <=
                                                               if (self._x <= uint144(-1)) {
138
    type(uint144).max) { // Changed here
```

```
139
                return
    uq112x112(uint224(Babylonian.sqrt(uint2
    56(self._x) << 112)));
140
            }
141
142
            uint8 safeShiftBits = 255 -
    BitMath.mostSignificantBit(self._x);
            safeShiftBits -= safeShiftBits
143
    % 2;
144
            return
    uq112x112(uint224(Babylonian.sqrt(uint2
    56(self._x) << safeShiftBits) << ((112
    - safeShiftBits) / 2)));
145
        }
146 }
147
```

```
139
                return
    uq112x112(uint224(Babylonian.sqrt(uint2
    56(self._x) << 112)));
140
            }
141
            uint8 safeShiftBits = 255 -
142
    BitMath.mostSignificantBit(self._x);
            safeShiftBits -= safeShiftBits
143
    % 2;
144
            return
    uq112x112(uint224(Babylonian.sqrt(uint2
    56(self._x) << safeShiftBits) << ((112
    - safeShiftBits) / 2)));
145
        }
146 }
147
```



### Comparing sensitive data, confidential files or internal emails?

Most legal and privacy policies prohibit uploading sensitive data online. Diffchecker Desktop ensures your confidential information never leaves your computer. Work offline and compare documents securely.

14

15

16

17

18

19

20

21

**Download Diffchecker Desktop** 

Learn more

# Untitled diff

```
8 removals
                                      52 lines
 1 // SPDX-License-Identifier: CC-BY-4.0
 2 pragma solidity >=0.4.0;
 3
 4 // Taken from
   https://medium.com/coinmonks/math-in-
   solidity-part-3-percents-and-
   proportions-4db014e080b1
 5 // License is CC-BY-4.0
 6 library FullMath {
       function fullMul(uint256 x, uint256
   y) internal pure returns (uint256 l,
   uint256 h) {
            uint256 mm = mulmod(x, y,
   type(uint256).max);
 9
            1 = x * y;
            h = mm - 1;
10
            if (mm < 1) h -= 1;
11
12
       }
13
       function fullDiv(
14
15
            uint256 1,
            uint256 h,
16
            uint256 d
17
        ) private pure returns (uint256) {
18
19
            uint256 pow2 = d \& (\sim d + 1); //
   Changed here
20
            d /= pow2;
21
            1 /= pow2;
```

```
1 // SPDX-License-Identifier: CC-BY-4.0
 2 pragma solidity >=0.4.0;
 3
 4 // taken from
   https://medium.com/coinmonks/math-in-
   solidity-part-3-percents-and-
   proportions-4db014e080b1
 5 // license is CC-BY-4.0
 6 library FullMath {
       function fullMul(uint256 x, uint256
 7
   y) internal pure returns (uint256 l,
   uint256 h) {
           uint256 mm = mulmod(x, y,
   uint256(-1));
           1 = x * y;
 9
           h = mm - 1;
10
           if (mm < 1) h -= 1;
11
12
       }
13
       function fullDiv(
```

uint256 1,

uint256 h,

uint256 d

d /= pow2;

1 /= pow2;

) private pure returns (uint256) {

uint256 pow2 = d & -d;

+ 6 additions

52 lines

```
//1 += h * ((-pow2 / pow2) +
   1); // Changed here, Just commenting it
   is useless and out of our use
23
            uint256 r = 1;
                                                    23
24
            r *= 2 - d * r;
                                                    24
25
            r *= 2 - d * r;
                                                    25
            r *= 2 - d * r;
26
                                                    26
            r *= 2 - d * r;
27
                                                    27
            r *= 2 - d * r;
28
                                                    28
            r *= 2 - d * r;
29
                                                    29
30
            r *= 2 - d * r;
                                                    30
            r *= 2 - d * r;
31
                                                    31
            return 1 * r;
32
                                                    32
33
                                                    33
        }
                                                            }
34
                                                    34
35
        function mulDiv(
                                                    35
            uint256 x,
36
                                                    36
37
            uint256 y,
                                                    37
38
            uint256 d
                                                    38
39
        ) internal pure returns (uint256) {
                                                    39
            (uint256 l, uint256 h) =
40
                                                    40
   fullMul(x, y);
                                                    41
41
            uint256 mm = mulmod(x, y, d);
42
                                                    42
            if (mm > 1) h -= 1;
                                                    43
43
            1 -= mm;
44
                                                    44
45
                                                    45
            if (h == 0) return 1 / d;
46
                                                    46
47
                                                    47
48
            require(h < d, 'FullMath:</pre>
                                                    48
   FULLDIV_OVERFLOW');
            return fullDiv(l, h, d);
49
                                                    49
       }
                                                            }
50
                                                    50
51 }
                                                    51 }
52
                                                    52
```

```
uint256 r = 1;
        r *= 2 - d * r;
        return 1 * r;
    function mulDiv(
        uint256 x,
        uint256 y,
        uint256 d
    ) internal pure returns (uint256) {
        (uint256 l, uint256 h) =
fullMul(x, y);
        uint256 mm = mulmod(x, y, d);
        if (mm > 1) h -= 1;
        1 -= mm;
        if (h == 0) return 1 / d;
        require(h < d, 'FullMath:</pre>
FULLDIV_OVERFLOW');
        return fullDiv(l, h, d);
```

1 += h \* ((-pow2) / pow2 + 1);