Platform Developer’s Kit

Fixed-point Library Manual
Contents

1 FIXED-POINT LIBRARY ................................................................. 4
  1.1 FIXED-POINT NOTATION .......................................................... 4
  1.2 FIXED-POINT LIBRARY HEADER FILE ........................................ 4
    1.2.1 FIXED_ISSIGNED ................................................................. 5
    1.2.2 FIXED_ISUNSIGNED ............................................................. 6
  1.3 FIXED-POINT STRUCTURES ...................................................... 6
    1.3.1 Signed fixed-point structures ............................................... 6
    1.3.2 Unsigned fixed-point structures ........................................... 10
  1.4 FIXED POINT FUNCTIONS ....................................................... 13
    1.4.1 FixedAbs ................................................................. 13
    1.4.2 FixedAdd ................................................................. 14
    1.4.3 FixedAnd ................................................................. 15
    1.4.4 FixedCastSigned ............................................................ 16
    1.4.5 FixedCastUnsigned ......................................................... 17
    1.4.6 FixedDivSigned .............................................................. 19
    1.4.7 FixedDivUnsigned ............................................................ 20
    1.4.8 FixedEq ................................................................. 21
    1.4.9 FixedFracWidth ............................................................... 22
    1.4.10 FixedGT ................................................................. 23
    1.4.11 FixedGTE ................................................................. 24
    1.4.12 FixedIntWidth ............................................................... 25
    1.4.13 FixedLeftShift ............................................................... 26
    1.4.14 FixedLiteral ................................................................. 27
    1.4.15 FixedLiteralFromInts ....................................................... 29
    1.4.16 FixedLT ................................................................. 30
    1.4.17 FixedLTE ................................................................. 31
    1.4.18 FixedMultSigned ............................................................ 32
    1.4.19 FixedMultUnsigned ......................................................... 33
    1.4.20 FixedNeg ................................................................. 34
    1.4.21 FixedNEq ................................................................. 35
    1.4.22 FixedNot ................................................................. 36
    1.4.23 FixedOr ................................................................. 37
    1.4.24 FixedRightShift ............................................................. 38
    1.4.25 FixedSub ................................................................. 40
    1.4.26 FixedToBits ................................................................. 41
    1.4.27 FixedToBool ................................................................. 42
    1.4.28 FixedToInt ................................................................. 43
    1.4.29 FixedXor ................................................................. 44

2 INDEX .................................................................................. 45
Conventions

A number of conventions are used in this document. These conventions are detailed below.

**Warning Message.** These messages warn you that actions may damage your hardware.

**Handy Note.** These messages draw your attention to crucial pieces of information.

Hexadecimal numbers will appear throughout this document. The convention used is that of prefixing the number with '0x' in common with standard C syntax.

Sections of code or commands that you must type are given in typewriter font like this:

```c
void main();
```

Information about a type of object you must specify is given in italics like this:

```c
copy SourceFileName DestinationFileName
```

Optional elements are enclosed in square brackets like this:

```c
struct [type_Name]
```

Curly brackets around an element show that it is optional but it may be repeated any number of times.

```c
string ::= "{character}"
```
**Assumptions & Omissions**

This manual assumes that you:

- have used Handel-C or have the Handel-C Language Reference Manual
- are familiar with common programming terms (e.g. functions)
- are familiar with MS Windows

This manual does not include:

- instruction in VHDL or Verilog
- instruction in the use of place and route tools
- tutorial example programs. These are provided in the Handel-C User Manual
1 Fixed-point library

The Fixed-point Library is installed as a library (.hcl) file with a header (.hch) file. The library is not board or device specific.

The fixed.hch header file must be included at the start of your program. It provides macro prototype declarations and preprocessor definitions. The functionality is stored in the fixed.hcl library file that must be added to your project within the DK GUI. This is done on the Project Settings dialog: Select the Linker tab and then type 'fixed.hcl ' in the Object/library modules box.

Handel-C libraries and header files previously used the .lib and .h extensions. These are no longer supported.

To use the library, you must first define a structure to hold the fixed-point number. Fixed-point numbers are represented as signed or unsigned structures.

1.1 Fixed-point notation

Mathematical notation (as in a decimal coinage system) in which the point separating whole numbers and fractions is in a fixed position.

1.2 Fixed-point library header file

To use the fixed-point library the header file fixed.hch needs to be included at the top of your program.

Previous versions of Handel-C used .h as the extension for header files. These are no longer supported.
Example

```c
#include <fixed.hch>
set clock = external "P1";
typedef FIXED_SIGNED(4, 4) MyFixed;

void main(void)
{
    MyFixed fixed1, fixed2;
    // Give the fixed-point number value -0.75
    fixed1 = FixedLiteral(FIXED_ISSIGNED, 4, 4, -1, 4);
    // Shift this number right by 1 bit
    fixed2 = FixedRightShift(fixed1, 1);
}
```

### 1.2.1 FIXED_ISSIGNED

**FIXED_ISSIGNED**

**Description**

This is a constant defined as 1. It is used to specify that a fixed-point value is signed. It does not define a FIXED_SIGNED structure.

**Requirements**

**Header file:** fixed.hch

**Example**

This definition is for use with FixedLiteral, FixedLiteralFromInts or casting:

```c
#include "fixed.hch"
set clock = external "P1";
typedef FIXED_SIGNED(4, 4) MyFixedSigned;
typedef FIXED_UNSIGNED(4, 4) MyFixedUnsigned;
void main(void)
{
    MyFixedSigned fixed1;
    MyFixedUnsigned fixed2;
    // Assign the value 1.25 to a signed fixed-point structure
    fixed1 = FixedLiteral(FIXED_ISSIGNED, 4, 4, 1.25);
    // Cast to unsigned
    fixed2 = FixedCastSigned(FIXED_ISUNSIGNED, 4, 4, fixed1);
}
```

The result is still 1.25 but stored as a signed fixed-point number:
fixed2.FixedIntBits = 1;
fixed2.FixedFracBits = 4;

### 1.2.2 FIXED_ISUNSIGNED

**FIXED_ISUNSIGNED**

**Description**

This is a constant defined as 0. It is used to specify that a fixed-point value is unsigned. It does not define a FIXED_UNSIGNED structure.

**Requirements**

Header file: fixed.hch

**Example**

This definition is for use with FixedLiteral, FixedLiteralFromInts or casting:

```c
#include <fixed.hch>
set clock = external "P1";
typedef FIXED_UNSIGNED(4, 8) MyFixed;
void main(void)
{
    MyFixed fixed1;
    // Assign the value 4.5 to a signed fixed-point structure
    fixed1 = FixedLiteral(FIXED_ISUNSIGNED, 4, 8, 4.5);
}
```

The result is 4.5 stored as a signed fixed-point number:

```c
fixed2.FixedIntBits = 4;
fixed2.FixedFracBits = 128;
```

## 1.3 Fixed-point structures

### 1.3.1 Signed fixed-point structures

To use signed fixed-point numbers in the fixed-point library, first you must define a structure with the following definition:

```c
FIXED_SIGNED(intWidth, fracWidth) Fixed;
```
This sets the width of the integer part of the number \textit{intWidth} and the width of the fraction part of the number \textit{fracWidth}. These widths can be any positive number including zero but must be compile time constants.

The definition creates a structure of the form:

```c
struct {
    signed intWidth FixedIntBits;
    signed fracWidth FixedFracBits;
};
```

The integer part of the number has width \textit{intWidth} and is held in: \textit{Fixed}.FixedIntBits

The fraction part of the number has width \textit{fracWidth} and is held in: \textit{Fixed}.FixedFracBits

**FIXED\_SIGNED**

```c
FIXED\_SIGNED( intWidth, fracWidth );
```

**Arguments**

\textit{intWidth} Width of integer part of the fixed-point structure. Must be positive and a compile time constant.

\textit{fracWidth} Width of fraction part of the fixed-point structure. Must be positive and a compile time constant.

**Return values**

None.

**Description**

Defines a structure to hold a signed fixed-point number of the type required for the other functions in the library. The structure takes the form:

```c
struct {
    signed intWidth FixedIntBits;
    signed fracWidth FixedFracBits;
};
```
Requirements

Header file: fixed.hch

Example

The suggested method of usage is to use this definition to create a type definition as follows:

typedef FIXED_SIGNED(intWidth, fracWidth) MyFixed;
MyFixed fixed1, fixed2;

Converting positive numbers to signed fixed-point structures

To convert a positive number to a fixed point structure you must define a FIXED_SIGNED structure to contain the number, and then assign it values using the FixedLiteral function.

Example

This shows how to define a 6-bit fixed-point number with 4 integer bits and 2 fraction bits, and then assign the value 3.5 to it.

#include <fixed.hch>
set clock = external "P1";
// Define a name for the structure type
typedef FIXED_SIGNED(4, 2) MyFixed;
void main(void)
{
    // Declare variable of type
    MyFixed fixedNumber;
    // Give fixedNumber the value 3.5
    fixedNumber = FixedLiteral(FIXED_ISSIGNED, 4, 2, 3.5);
}

This is stored as

fixedNumber.FixedIntBits = 3
fixedNumber.FixedFracBits = 2

Explanation

FixedLiteral( isSigned , intWidth , fracWidth , floatConst );
returns a signed fixed-point number if isSigned is 1 or an unsigned fixed-point number if isSigned is 0. The number has the value floatConst with an integer part of width intWidth and a fraction part of width fracWidth.

The FixedLiteral function

- Sets Fixed.fixedIntBits to the value of the integer part of the number.
- Finds 2 to the power of the number of fraction bits, fracWidth.
• Multiplies this with the decimal part of the number.
• Rounds to the nearest integer and set Fixed.FixedFracBits to this value.

For a number \(a\cdot b\) the formulae are:
• \(\text{Fixed}.\text{FixedIntBits} = a\)
• \(\text{Fixed}.\text{FixedFracBits} = \text{integer part of } (b \times 2^{\text{fracWidth}})\)

**Converting negative numbers to signed fixed-point structures**

To convert a negative number to a fixed point structure you must define a FIXED_SIGNED structure to contain the number, and then assign it values using the FixedLiteral function.

**Example**

This shows how to define an 8 bit fixed-point number with 4 integer bits and 4 fraction bits, and then assign the value -3.5 to it.

```c
#include <fixed.hch>
set clock = external "P1";
// Define a name for the structure type
typedef FIXED_SIGNED(4, 4) MyFixed;
void main(void)
{
    // Declare variable of type
    MyFixed fixedNumber;
    // Give fixedNumber the value -3.5
    fixedNumber = FixedLiteral( FIXED_ISSIGNED, 4, 4, -3.5);
}
```

This is stored as

\[\text{fixedNumber}.\text{FixedIntBits} = -4\]
\[\text{fixedNumber}.\text{FixedFracBits} = 8\]

**Explanation**

FixedLiteral( isSigned , intWidth , fracWidth , floatConst );
returns a signed fixed-point number if isSigned is 1 or an unsigned fixed-point number if isSigned is 0. The number has the value floatConst with an integer part of width intWidth and a fraction part of width fracWidth.

The FixedLiteral function

• Sets Fixed.FixedIntBits to the value of the integer part of the number.
• Finds 2 to the power of the number of fraction bits, fracWidth, takes the decimal part of the number from 1 and multiplies them together
• Rounds to the nearest integer and set \( \text{Fixed}.\text{FixedFracBits} \) to this value.
• If \( \text{Fixed}.\text{FixedFracBits} \) is zero then does not change \( \text{Fixed}.\text{FixedIntBits} \).
• If \( \text{Fixed}.\text{FixedFracBits} \) is not zero takes 1 from \( \text{Fixed}.\text{FixedIntBits} \).

For a number \( a.b \) the formulae are:
• \( \text{Fixed}.\text{FixedFracBits} = \) integer part of \( ((1-b) \times 2^{\text{fracWidth}}) \)
• If \( \text{Fixed}.\text{FixedFracBits} \) is zero: \( \text{Fixed}.\text{FixedIntBits} = a \)
• Else if \( \text{Fixed}.\text{FixedFracBits} \) is not zero: \( \text{Fixed}.\text{FixedIntBits} = a-1 \)

1.3.2 Unsigned fixed-point structures

To use unsigned fixed-point numbers in the fixed-point library, first you must define a structure with the following definition:

\[
\text{FIXED_UNSIGNED}(\text{intWidth}, \text{fracWidth}) \text{ Fixed};
\]

This sets the width of the integer part of the number \( \text{intWidth} \) and the width of the fraction part of the number \( \text{fracWidth} \). These widths can be any positive number including zero but must be compile time constants.

The definition creates a structure of the form:

\[
\text{struct}
\{
    \text{unsigned} \ \text{intWidth} \ \text{FixedIntBits};
    \text{unsigned} \ \text{fracWidth} \ \text{FixedFracBits};
\};
\]

The integer part of the number has width \( \text{intWidth} \) and is held in \( \text{Fixed}.\text{FixedIntBits} \).

The fraction part of the number has width \( \text{fracWidth} \) and is held in: \( \text{Fixed}.\text{FixedFracBits} \)

\[
\text{FIXED UNSIGNED}
\]

\[
\text{FIXED UNSIGNED}( \text{intWidth}, \text{fracWidth} );
\]
Arguments

**intWidth**
Width of integer part of the fixed-point structure. Must be positive and a compile time constant.

**fracWidth**
Width of fraction part of the fixed-point structure. Must be positive and a compile time constant.

Return values
None.

Description
Defines a structure to hold an unsigned fixed-point number of the type required for the other functions in the library. The structure takes the form:

```
struct
{
    unsigned intWidth FixedIntBits;
    unsigned fracWidth FixedFracBits;
};
```

Requirements
Header file: fixed.hch

Example
The suggested method of usage is to use this definition to create a type definition as follows:

```
typedef FIXED_UNSIGNED(intWidth, fracWidth) MyFixed;
MyFixed fixed1, fixed2;
```

Converting unsigned numbers to unsigned fixed-point structures
To convert a number to an unsigned fixed point structure you must define a FIXED_UNSIGNED structure to contain the number, and then assign it values using the FixedLiteral function.

Example
To define an 8 bit fixed-point number with 4 integer bits and 4 fraction bits, and then assign the value 10.5 to it.
```c
#include <fixed.h>
set clock = external "P1";
// Define a name for the structure type
typedef FIXED_UNSIGNED(4, 4) MyFixed;
void main(void)
{
    // Declare variable of type
    MyFixed fixedNumber;
    // Give fixedNumber the value 10.5
    fixedNumber = FixedLiteral( FIXED_ISUNSIGNED, 4, 4, 10.5);
}
This is stored as:
fixedNumber.FixedIntBits = 10
fixedNumber.FixedFracBits = 8
```

**Explanation**

`FixedLiteral(isSigned, intWidth, fracWidth, floatConst);`
returns a signed fixed-point number if `isSigned` is 1 or an unsigned fixed-point number if `isSigned` is 0. The number has the value `floatConst` with an integer part of width `intWidth` and a fraction part of width `fracWidth`.

The `FixedLiteral` function

- Sets `Fixed.fixedIntBits` to the value of the integer part of the number.
- Finds 2 to the power of the number of fraction bits, `fracWidth`.
- Multiplies this with the decimal part of the number.
- Rounds to the nearest integer and set `Fixed.FixedFracBits` to this value.

For a number `a.b` the formulae are:

- `Fixed.FixedIntBits = a`
- `Fixed.FixedFracBits = integer part of (b \times 2^{fracWidth})`
1.4 Fixed point functions

1.4.1 FixedAbs

FixedAbs( Fixed );

Arguments
Fixe Fixed-point structure of signed type
d and any width

Return values
Fixed-point number of signed type and same width as Fixed.

Description
Returns the absolute value of Fixed. The number returned is of the same width as Fixed so any bits outside this width are lost. Signed integers use 2’s complement representation in Handel-C so

abs(max positive number) < abs(min negative number)

This means the function gives the result:

abs(min negative number) = min negative number.

Requirements
Header file: fixed.hch
Library file: fixed.hcl

Example
This example shows finding the absolute value of a FIXED_SIGNED(4, 4).

#include <fixed.hch>
set clock = external "P1";
typedef FIXED_SIGNED(4, 4) MyFixed;

void main(void)
{
    MyFixed fixed1, fixed2;
    // Give the fixed-point number value -7.25
    fixed1 = FixedLiteral(FIXED_ISSIGNED, 4, 4, -7.25);
    // Find the absolute value of this number
    fixed2 = FixedAbs(fixed1);
}

The result is 7.25. This is stored as:
fixed2.FixedIntBits = 7;
fixed2.FixedFracBits = 4;

### 1.4.2 FixedAdd

FixedAdd( Fixed1, Fixed2 );

**Arguments**

- **Fixed1** Fixed-point structure of any type and width
- **Fixed2** Fixed-point structure of the same type and width

**Return values**

Fixed-point number of the same type and width as **Fixed1** and **Fixed2**.

**Description**

Returns **Fixed1** added to **Fixed2**. The number returned is of the same width as **Fixed1** so any bits outside this width are lost.

**Requirements**

- Header file: fixed.hch
- Library file: fixed.hcl

**Example**

This example shows addition on two **FIXED_UNSIGNED(4, 8)**.

```c
#include <fixed.hch>
set clock = external "P1";
typedef FIXED_UNSIGNED(4, 8) MyFixed;

void main(void)
{
    MyFixed fixed1, fixed2, fixed3;
    // Give the fixed-point number value 3.25
    fixed1 = FixedLiteralFromInts(FIXED_IS_UNSIGNED, 4, 8, 3, 64);
    // Give the fixed-point number value 4.75
    fixed2 = FixedLiteralFromInts(FIXED_IS_UNSIGNED, 4, 8, 4, 192);
    // Add the numbers together
    fixed3 = FixedAdd(fixed1, fixed2);
}
```

The result is 8. This is stored as:
fixed3.FixedIntBits = 8;
fixed3.FixedFracBits = 0;

### 1.4.3 FixedAnd

**FixedAnd** *(Fixed1, Fixed2)*;

**Arguments**

- **Fixed 1**: Fixed-point structure of any type and width
- **Fixed 2**: Fixed-point structure of the same type and width

**Return values**

Fixed-point number of the same type and width as *Fixed1* and *Fixed2*.

**Description**

Returns bitwise AND of *Fixed1* and *Fixed2*.

**Requirements**

- Header file: fixed.hch
- Library file: fixed.hcl

**Example**

This example finds the bitwise AND of two *FIXED_UNSIGNED(0, 16)*.

```c
#include <fixed.hch>
set clock = external "P1";
typedef FIXED_UNSIGNED(0, 16) MyFixed;

void main(void)
{
    MyFixed fixed1, fixed2, fixed3;
    // Give the fixed-point number value 0.02734375
    fixed1 = FixedLiteralFromInts(FIXED_ISUNSIGNED, 0, 16, 0, 1792);
    // Give the fixed-point number value 0.234375
    fixed2 = FixedLiteralFromInts(FIXED_ISUNSIGNED, 0, 16, 0, 15360);
    // And these numbers
    fixed3 = FixedAnd(fixed1, fixed2);
}
```

The result is 0.015625. This is stored as:
fixed3.FixedIntBits = 0;
fixed3.FixedFracBits = 1024;

1.4.4 FixedCastSigned

FixedCastSigned( isSigned, intWidth, fracWidth, Fixed );

If you need to cast from a signed number to a fixed-point signed or fixed-point unsigned number, use FixedCastSigned. If you need to cast from an unsigned number, use FixedCastUnsigned.

Arguments

isSigned Compile time constant to indicate the type of fixed-point structure. FIXED_ISSIGNED represents signed and FIXED_ISUNSIGNED represents unsigned.

intWidth Width of integer part of the fixed-point structure. Must be positive and a compile time constant.

fracWidth Width of fraction part of the fixed-point structure. Must be positive and a compile time constant.

Fixed Fixed-point structure of signed type and any width.

Return values

Fixed-point structure of the type and width specified.

Description

Casts any signed fixed-point number to the type and width specified. Any bits added will be sign extended and any bits lost will be truncated.

Requirements

Header file: fixed.hch

Library file: fixed.hcl

Example

This example casts a FIXED_SIGNED(4, 4) to a FIXED_UNSIGNED(4, 4).

#include <fixed.hch>
set clock = external "P1";
typedef FIXED_SIGNED(4, 4) MyFixedSigned;
typedef FIXED_UNSIGNED(4, 4) MyFixedUnsigned;

void main(void)
{
    MyFixedSigned fixed1;
    MyFixedUnsigned fixed2;
    // Assign the value 7.125
    fixed1 = FixedLiteral(FIXED_ISSIGNED, 4, 4, 7.125);
    // Cast to unsigned
    fixed2 = FixedCastSigned(FIXED_ISUNSIGNED, 4, 4, fixed1);
}

The result is still 7.125 but stored as a signed fixed-point number:

fixed2.FixedIntBits = 7;
fixed2.FixedFracBits = 2;

### 1.4.5 FixedCastUnsigned

FixedCastUnsigned( isSigned, intWidth, fracWidth, Fixed);

If you need to cast from an unsigned number to a fixed-point signed or fixed-point unsigned number, use FixedCastUnsigned. If you need to cast from a signed number, use FixedCastSigned.

**Arguments**

- **isSigned** Compile time constant to indicate the type of fixed-point structure. FIXED_ISSIGNED represents signed and FIXED_ISUNSIGNED represents unsigned.

- **intWidth** Width of integer part of the fixed-point structure. Must be positive and a compile time constant.

- **fracWidth** Width of fraction part of the fixed-point structure. Must be positive and a compile time constant.

- **Fixed** Fixed-point structure of unsigned type and any width.

**Return values**

Fixed-point structure of the type and width specified.
**Description**
Casts any unsigned fixed-point number to the type and width specified. Any bits added will 0 and any bits lost will be truncated.

**Requirements**
- **Header** file: fixed.hch
- **Library** file: fixed.hcl

**Example**
This example casts a FIXED_UNSIGNED(4, 4) to a FIXED_UNSIGNED(16, 16).

```c
#include <fixed.hch>
set clock = external "P1";
typedef FIXED_UNSIGNED(4, 4) MyFixedSmall;
typedef FIXED_UNSIGNED(16, 16) MyFixedBig;

void main(void)
{
    MyFixedSmall fixed1;
    MyFixedBig fixed2;
    // Assign the value 15.5
    fixed1 = FixedLiteral(FIXED_ISUNSIGNED, 4, 4, 15.5);
    // Cast to the larger width
    fixed2 = FixedCastUnsigned(FIXED_ISUNSIGNED, 16, 16, fixed1);
}
```

The result is still 15.5 but stored as a fixed-point number with a different width:

```c
fixed2.FixedIntBits = 15;
fixed2.FixedFracBits = 32768;
```
1.4.6 FixedDivSigned

FixedDivSigned( Fixed1, Fixed2 );

**Arguments**

*Fixed1*  Fixed-point structure of signed type
and any width

*Fixed2*  Fixed-point structure of signed type
and the same width

**Return values**

Fixed-point number of signed type and the same width as *Fixed1* and *Fixed2*.

**Description**

Divisor for signed fixed-point numbers only. Returns *Fixed1* divided by *Fixed2*. The number returned is of the same width as *Fixed1* so any bits outside this width are lost.

**Requirements**

Header file:  fixed.hch
Library file:  fixed.hcl

**Example**

This example shows division on FIXED_SIGNED(4, 4).

```
#include <fixed.hch>
set clock = external "P1";
typedef FIXED_SIGNED(4, 4) MyFixed;

void main(void)
{
    MyFixed fixed1, fixed2, fixed3;
    // Give the fixed-point number value -5
    fixed1 = FixedLiteral(FIXED_ISSIGNED, 4, 4, -5);
    // Give the fixed-point number value 4
    fixed2 = FixedLiteral(FIXED_ISSIGNED, 4, 4, 4);
    // Divide these numbers
    fixed3 = FixedDivSigned(fixed1, fixed2);
}
```

The result is 1.25. This is stored as:

```
fixed3.FixedIntBits = -2;
fixed3.FixedFracBits = -4;
```
1.4.7 FixedDivUnsigned

FixedDivUnsigned( Fixed1, Fixed2 );

Arguments

Fixed 1
Fixed-point structure of unsigned type
and any width

Fixed 2
Fixed-point structure of unsigned type
and the same width

Return values

Fixed-point number of unsigned type and the same width as Fixed1 and Fixed2.

Description

Divisor for unsigned fixed-point numbers only. Returns Fixed1 divided by Fixed2. The number returned is of the same width as Fixed1 so any bits outside this width are lost.

Requirements

Header file: fixed.hch
Library file: fixed.hcl

Example

This example shows division on FIXED_UNSIGNED(4, 4).

#include <fixed.hch>
set clock = external "P1";
typedef FIXED_UNSIGNED(4, 4) MyFixed;

void main(void)
{
    MyFixed fixed1, fixed2, fixed3;
    // Give the fixed-point number value 15
    fixed1 = FixedLiteral(FIXED_IS_UNSIGNED, 4, 4, 15);
    // Give the fixed-point number value 2
    fixed2 = FixedLiteral(FIXED_IS_UNSIGNED, 4, 4, 2);
    // Multiply these numbers
    fixed3 = FixedDivUnsigned(fixed1, fixed2);
}

The result is 7.5. This is stored as:

fixed3.FixedIntBits = 7;
fixed3.FixedFracBits = 8;
1.4.8 FixedEq

FixedEq( Fixed1, Fixed2 );

Arguments

Fixed 1 Fixed-point structure of any type
Fixed 2 Fixed-point structure of the same type and width

Return values

Single bit wide integer with 0 as false and 1 as true.

Description

Returns true if Fixed1 equals Fixed2.

Requirements

Header file: fixed.hch
Library file: fixed.hcl

Example

This example tests the equality of two FIXED_UNSIGNED(16, 16).

#include <fixed.hch>
set clock = external "P1";
typedef FIXED_UNSIGNED(16, 16) MyFixed;

void main(void)
{
    unsigned int 1 result;
    MyFixed fixed1, fixed2;
    // Give the fixed-point number value 1000.046875
    fixed1 = FixedLiteralFromInts(FIXED_IS_UNSIGNED, 16, 16, 1000, 3072);
    // Give the fixed-point number value 1000.03125
    fixed2 = FixedLiteralFromInts(FIXED_IS_UNSIGNED, 16, 16, 1000, 2048);
    // Are these numbers equal?
    result = FixedEq(fixed1, fixed2);
}

fixed1 is not equal to fixed2 so:

result = 0;
1.4.9 FixedFracWidth

FixedFracWidth( Fixed );

**Arguments**

*Fixed* Fixed-point structure of any type and width

d

**Return values**

Compile time constant integer.

**Description**

Returns width of the fraction part of *Fixed*.

**Requirements**

Header file: fixed.hch

Library file: fixed.hcl

**Example**

This example finds the width of the fraction part of a FIXED_SIGNED(16, 8).

```c
#include <fixed.hch>
set clock = external "P1";
typedef FIXED_SIGNED(16, 8) MyFixed;

void main(void)
{
    unsigned int 5 result;
    MyFixed fixed;
    // Find the width
    result = FixedFracWidth(fixed);
}
```

The result is 8.
1.4.10 FixedGT

FixedGT( Fixed1, Fixed2 );

Arguments

- **Fixed1** Fixed-point structure of any type and width
- **Fixed2** Fixed-point structure of the same type and width

Return values

Single bit wide integer with 0 as false and 1 as true.

Description

Returns true if Fixed1 is greater than Fixed2.

Requirements

- **Header file:** fixed.hch
- **Library file:** fixed.hcl

Example

This example tests for greater than of two FIXED_SIGNED(4, 16).

```c
#include <fixed.hch>
set clock = external "P1";
t typedef FIXED_SIGNED(4, 16) MyFixed;

void main(void)
{
    unsigned int 1 result;
    MyFixed fixed1, fixed2;
    // Give the fixed-point number value 5.125
    fixed1 = FixedLiteral(FIXED_ISSIGNED, 4, 16, 5.125);
    // Give the fixed-point number value -5.125
    fixed2 = FixedLiteral(FIXED_ISSIGNED, 4, 16, -5.125);
    // Is fixed1 > fixed2
    result = FixedGT(fixed1, fixed2);
}
```

**fixed1** is greater than **fixed2** so:

result = 1;
### 1.4.11 FixedGTE

FixedGTE( Fixed1, Fixed2 );

**Arguments**

*Fixed1*  
Fixed-point structure of any type and width

*Fixed2*  
Fixed-point structure of the same type and width

**Return values**

Single bit wide integer with 0 as false and 1 as true.

**Description**

Returns true if *Fixed1* is greater than or equal to *Fixed2*.

**Requirements**

- **Header file:** fixed.hch
- **Library file:** fixed.hcl

**Example**

This example tests for greater than or equal to of two FIXED_SIGNED(4, 16).

```c
#include <fixed.hch>
set clock = external "P1";
typedef FIXED_SIGNED(4, 16) MyFixed;

void main(void)
{
    unsigned int 1 result;
    MyFixed fixed1, fixed2;
    // Give the fixed-point number value 5.125
    fixed1 = FixedLiteral(FIXED_ISSIGNED, 4, 16, 5.125);
    // Give the fixed-point number value -5.125
    fixed2 = FixedLiteral(FIXED_ISSIGNED, 4, 16, -5.125);
    // Is fixed1 > fixed2
    result = FixedGTE(fixed1, fixed2);
}

fixed1 is greater than or equal to fixed2 so:
result = 1;
```
1.4.12 FixedIntWidth

FixedIntWidth( Fixed );

Arguments

Fixed Fixed-point structure of any type and width

Return values

Compile time constant integer.

Description

Returns width of the integer part of Fixed.

Requirements

Header

fixed.hch

file:

Library

fixed.hcl

file:

Example

This example finds the width of the integer part of a FIXED_SIGNED(16, 8).

#include <fixed.hch>
set clock = external "P1";
typedef FIXED_SIGNED(16, 8) MyFixed;

void main(void)
{
    unsigned int 5 result;
    MyFixed fixed;
    // Find the width
    result =FixedIntWidth(fixed);
}

The result is 16.
### 1.4.13 **FixedLeftShift**

**FixedLeftShift**(  *Fixed*,  *Shift* );

**Arguments**

*Fixed*  
Fixed-point structure of any type and width

*Shift*  
Number of bits to shift left by

**Return values**

Fixed-point number of same type and width as *Fixed*.

**Description**

Returns *Fixed* shifted left by *Shift* number of bits. The number returned is of the same width as *Fixed* so any bits shifted outside this width are lost.

The *Shift* expression must be unsigned and of width

\[ \text{shiftWidth} = \log_2 \left( \text{intWidth} + \text{fracWidth} + 1 \right) \]

where *intWidth* is width(*Fixed*.FixedIntBits) and *fracWidth* is width(*Fixed*.FixedFracBits) (as defined in FIXED_SIGNED and FIXED_UNSIGNED.)

*Shift* has the range: 0 to \(\exp(2^{\text{shiftWidth}}) - 1\).

If *Shift* is 0 no shift occurs. Shifts of \((\text{intWidth} + \text{fracWidth})\) or greater shift all the bits out of *Fixed* and produce a zero result.

**Requirements**

Header file: fixed.hch
Library file: fixed.hcl

**Example**

This example shows left shifting on a FIXED_UNSIGNED(8, 4). The integer part has value 9 and the fraction part has value \(\frac{1}{2}\).
#include <fixed.hch>
set clock = external "P1";
typedef FIXED_UNSIGNED(8, 4) MyFixed;

void main(void)
{
    MyFixed fixed1, fixed2;
    // Give the fixed-point number value 9.5
    fixed1 = FixedLiteralFromInts(FIXED_ISUNSIGNED, 8, 4, 9, 8);
    // Shift this number left by 2 bits
    fixed2 = FixedLeftShift(fixed1, 2);
}

The result is 39. This is stored as:
fixed2.FixedIntBits = 38;
fixed2.FixedFracBits = 0;

1.4.14 FixedLiteral

FixedLiteral( isSigned, intWidth, fracWidth, floatConst );

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isSigned</td>
<td>Single bit wide unsigned integer with FIXED_ISSIGNED denoting signed and FIXED_ISUNSIGNED indicating unsigned. Must be a compile time constant.</td>
</tr>
<tr>
<td>intWidth</td>
<td>Width of integer part of the fixed-point structure. Must be positive and a compile time constant.</td>
</tr>
<tr>
<td>fracWidth</td>
<td>Width of fraction part of the fixed-point structure. Must be positive and a compile time constant.</td>
</tr>
<tr>
<td>floatConst</td>
<td>Floating-point constant value to assign value of fixed-point structure.</td>
</tr>
</tbody>
</table>

Return values

Fixed-point number of the type and width specified.

Description

Returns a signed fixed-point number if isSigned is FIXED_ISSIGNED or an unsigned fixed-point number if isSigned is FIXED_ISUNSIGNED. The number has the value
floatConst with an integer part of width intWidth and a fraction part of width fracWidth.

Requirements

Header file: fixed.hch
Library file: fixed.hcl

Example 1:
This example assigns values to a FIXED_SIGNED(16, 8).

typedef FIXED_SIGNED(16, 8) MyFixed;
void main(void)
{
    MyFixed fixed;
    // Assign the value 32767.5
    fixed = FixedLiteral(FIXED_ISSIGNED, 16, 8, 32767.5);
}

This gives the structure the values:
fixed.FixedIntBits = 32767;
fixed.FixedFracBits = 128;

Example 2:
This example assigns values to a FIXED_UNSIGNED(16, 8).

typedef FIXED_UNSIGNED(16, 8) MyFixed;
void main(void)
{
    MyFixed fixed;
    // Assign the value 32767.5
    fixed = FixedLiteral(FIXED_ISUNSIGNED, 16, 8, 32767.5);
}

This gives the structure the values:
fixed.FixedIntBits = 37267;
fixed.FixedFracBits = 128;
1.4.15 FixedLiteralFromInts

FixedLiteralFromInts( isSigned, intWidth, fracWidth,
    intBits, fracBits );

Arguments

isSigned Single bit wide unsigned integer with
    FIXED_ISSIGNED denoting signed and
    FIXED_ISUNSIGNED indicating
    unsigned. Must be a compile time
    constant.

intWidth Width of integer part of the fixed-point
    structure. Must be positive and a
    compile time constant.

fracWidth Width of fraction part of the fixed-
    point structure. Must be positive and a
    compile time constant.

intBits Value to set to the integer part of the
    fixed-point structure. Must be of width
    intWidth.

fracBits Value to set to the fraction part of the
    fixed-point structure. Must be of width
    fracWidth.

Return values

Fixed-point number of the type and width specified.

Description

Returns a signed fixed-point number if isSigned is 1 or an unsigned fixed-point number
if isSigned is 0. The number has an integer part intBits of width intWidth and a
fraction part fracBits of width fracWidth.

Requirements

Header file: fixed.hch
Library file: fixed.hcl

Example 1:

This example assigns values to a FIXED_SIGNED(16, 8).
typedef FIXED_SIGNED(16, 8) MyFixed;
void main(void)
{
    MyFixed fixed;
    // Assign the value 32767.5
    fixed = FixedLiteralFromInts(FIXED_ISSIGNED, 16, 8, 32767, 128);
}

This gives the structure the values:
fixed.FixedIntBits = 32767;
fixed.FixedFracBits = 128;

Example 2:
This example shows assigns values to a FIXED_UNSIGNED(16, 8).

typedef FIXED_UNSIGNED(16, 8) MyFixed;
void main(void)
{
    MyFixed fixed;
    // Assign the value 32767.5
    fixed = FixedLiteralFromInts(FIXED_ISUNSIGNED, 16, 8, 37267, 128);
}

This gives the structure the values:
fixed.FixedIntBits = 37267;
fixed.FixedFracBits = 128;

1.4.16 FixedLT

FixedLT( Fixed1, Fixed2);

Arguments

Fixed1 Fixed-point structure of any type

Fixed2 Fixed-point structure of the same type and width

Return values

Single bit wide integer with 0 as false and 1 as true.

Description

Returns true if Fixed1 is less than Fixed2.
Requirements

Header file: fixed.hch
Library file: fixed.hcl

Example

This example tests for less than of two FIXED_UNSIGNED(4, 4).

```
#include <fixed.hch>
set clock = external "P1";
typedef FIXED_UNSIGNED(4, 4) MyFixed;

void main(void)
{
    unsigned int result;
    MyFixed fixed1, fixed2;
    // Give the fixed-point number value 3.5
    fixed1 = FixedLiteral(FIXED_ISUNSIGNED, 4, 4, 3.5);
    // Give the fixed-point number value 3.5
    fixed2 = FixedLiteral(FIXED_ISUNSIGNED, 4, 4, 3.5);
    // Is fixed1 < fixed2
    result = FixedLT(fixed1, fixed2);
}
```

fixed1 is not less than fixed2 so:

result = 0;

1.4.17 FixedLTE

FixedLTE( Fixed1, Fixed2 );

Arguments

<table>
<thead>
<tr>
<th>Fixed 1</th>
<th>Fixed-point structure of any type and width</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed 2</td>
<td>Fixed-point structure of the same type and width</td>
</tr>
</tbody>
</table>

Return values

Single bit wide integer with 0 as false and 1 as true.

Description

Returns true if Fixed1 is less than or equal to Fixed2.
Requirements

Header file: fixed.hch
Library file: fixed.hcl

Example

This example tests for less than or equal to of two FIXED_UNSIGNED(4, 4).

```c
#include <fixed.hch>
set clock = external "P1";
typedef FIXED_UNSIGNED(4, 4) MyFixed;

void main(void)
{
    unsigned int 1 result;
    MyFixed fixed1, fixed2;
    // Give the fixed-point number value 3.5
    fixed1 = FixedLiteralFromInts(FIXED_ISUNSIGNED, 4, 4, 3, 8);
    // Give the fixed-point number value 3.5
    fixed2 = FixedLiteralFromInts(FIXED_ISUNSIGNED, 4, 4, 3, 8);
    // Is fixed1 less than or equal to fixed2
    result = FixedLTE(fixed1, fixed2);
}
```

fixed1 is less than or equal to fixed2 so:
result = 1;

1.4.18 FixedMultSigned

FixedMultSigned( Fixed1, Fixed2 );

Arguments

Fixed1  Fixed-point structure of signed type and any width
Fixed2  Fixed-point structure of signed type and the same width

Return values

Fixed-point number of signed type and the same width as Fixed1 and Fixed2.

Description

Multiplier for signed fixed-point numbers only. Returns Fixed1 multiplied by Fixed2. The number returned is of the same width as Fixed1 so any bits outside this width are lost.
Requirements
Header file: fixed.hch
Library file: fixed.hcl

Example
This example shows multiplication on FIXED_SIGNED(1, 16).

```c
#include <fixed.hch>
set clock = external "P1";
typedef FIXED_SIGNED(1, 16) MyFixed;

void main(void)
{
    MyFixed fixed1, fixed2, fixed3;
    // Give the fixed-point number value -0.5
    fixed1 = FixedLiteral(FIXED_ISSIGNED, 1, 16, -0.5);
    // Give the fixed-point number value -0.125
    fixed2 = FixedLiteral(FIXED_ISSIGNED, 1, 16, -0.125);
    // Multiply these numbers
    fixed3 = FixedMultSigned(fixed1, fixed2);
}
```

The result is 0.0625. This is stored as:

```c
fixed3.FixedIntBits = 0;
fixed3.FixedFracBits = 4096;
```

1.4.19 `FixedMultUnsigned`

**FixedMultUnsigned( Fixed1, Fixed2 );**

**Arguments**

- `Fixed1` Fixed-point structure of unsigned type and any width
- `Fixed2` Fixed-point structure of unsigned type and the same width

**Return values**

Fixed-point number of unsigned type and the same width as `Fixed1` and `Fixed2`.

**Description**

Multiplier for unsigned fixed-point numbers only. Returns `Fixed1` multiplied by `Fixed2`. The number returned is of the same width as `Fixed1` so any bits outside this width are lost.
**Requirements**

**Header**  
file: fixed.hch

**Library**  
file: fixed.hcl

**Example**

This example shows multiplication on FIXED_UNSIGNED(1, 16).

```c
#include <fixed.hch>
set clock = external "P1";
typedef FIXED_UNSIGNED(1, 16) MyFixed;

void main(void)
{
    MyFixed fixed1, fixed2, fixed3;
    // Give the fixed-point number value 0.5
    fixed1 = FixedLiteral(FIXED_ISUNSIGNED, 1, 16, 0.5);
    // Give the fixed-point number value 0.125
    fixed2 = FixedLiteral(FIXED_ISUNSIGNED, 1, 16, 0.125);
    // Multiply these numbers
    fixed3 = FixedMultUnsigned(fixed1, fixed2);
}
```

The result is 0.0625. This is stored as:

```c
fixed3.FixedIntBits = 0;
fixed3.FixedFracBits = 4096;
```

### 1.4.20 FixedNeg

**FixedNeg( Fixed );**

**Arguments**

*Fixed*  
Fixed-point structure of signed type and any width

**Return values**

Fixed-point number of same type and width as *Fixed*.

**Description**

Returns the negative of *Fixed*. 
Requirements

Header file: fixed.hch
Library file: fixed.hcl

Example

This example negates a FIXED_SIGNED(4, 4).

```c
#include <fixed.hch>
set clock = external "P1";
typedef FIXED_SIGNED(4, 4) MyFixed;

void main(void)
{
    MyFixed fixed1, fixed2;
    // Give the fixed-point number value -1.625
    fixed1 = FixedLiteralFromInts(FIXED_ISSIGNED, 4, 4, -2, 6);
    // Find the negative of this number
    fixed2 = FixedNeg(fixed1);
}
```

The result is 1.625. This is stored as:

```c
defined2.FixedIntBits = 1;
defined2.FixedFracBits = 10;
```

1.4.21 FixedNEq

FixedNEq( Fixed1, Fixed2 );

Arguments

Fixed 1 Fixed-point structure of any type
Fixed 2 Fixed-point structure of the same type and width

Return values

Single bit wide integer with 0 as false and 1 as true.

Description

Returns true if Fixed1 does not equal Fixed2.
Requirements

Header file: fixed.hch
Library file: fixed.hcl

Example

This example tests for non-equality of two \texttt{FIXED\_UNSIGNED(16, 16)}.

```c
#include <fixed.hch>
set clock = external "P1";
typedef FIXED\_UNSIGNED(16, 16) MyFixed;

void main(void)
{
    unsigned int 1 result;
    MyFixed fixed1, fixed2;
    // Give the fixed-point number value 1000.046875
    fixed1 = FixedLiteral(FIXED\_ISUNSIGNED, 16, 16, 1000.046875);
    // Give the fixed-point number value 1000.03125
    fixed2 = FixedLiteral(FIXED\_ISUNSIGNED, 16, 16, 1000.03125);
    // Are these numbers not equal?
    result = FixedNEq(fixed1, fixed2);
}
```

\texttt{fixed1} is not equal to \texttt{fixed2} so:

```
result = 1;
```

\textbf{1.4.22 FixedNot}

\texttt{FixedNot( Fixed );}

\textbf{Arguments}

\textit{Fixed} Fixed-point structure of any type
\textit{d} and width

\textbf{Return values}

Fixed-point number of the same type and width as \textit{Fixed}.

\textbf{Description}

Returns bitwise NOT of \textit{Fixed}. 
Requirements

Header file:  fixed.hch  
Library file:  fixed.hcl  

Example

This example finds the bitwise NOT of a FIXED_SIGNED(4, 4).

```c
#include <fixed.hch>
set clock = external "P1";
typedef FIXED_SIGNED(4, 4) MyFixed;

void main(void)
{
    MyFixed fixed1, fixed2;
    // Give the fixed-point number value -5.875
    fixed1 = FixedLiteral(FIXED_ISSIGNED, 4, 4, -5.875);
    // Find the bitwise not of this number
    fixed2 = FixedNot(fixed1);
}
```

The result is 5.8125. This is stored as:

```c
fixed2.FixedIntBits = 5;
fixed2.FixedFracBits = 13;
```

1.4.23 FixedOr

FixedOr( Fixed1, Fixed2 );

Arguments

<table>
<thead>
<tr>
<th>Fixed</th>
<th>Fixed-point structure of any type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>and width</td>
</tr>
<tr>
<td>Fixed</td>
<td>Fixed-point structure of the same</td>
</tr>
<tr>
<td>2</td>
<td>type and width</td>
</tr>
</tbody>
</table>

Return values

Fixed-point number of the same type and width as Fixed1 and Fixed2.

Description

Returns bitwise inclusive OR of Fixed1 and Fixed2.
**Requirements**

Header file:  fixed.hch  
Library file:  fixed.hcl  

**Example**  
This example finds the bitwise OR of two FIXED_SIGNED(5, 5).

```c
#include <fixed.hch>
set clock = external "P1";
typedef FIXED_SIGNED(5, 5) MyFixed;

void main(void)
{
    MyFixed fixed1, fixed2, fixed3;
    // Give the fixed-point number value -1
    fixed1 = FixedLiteral(FIXED_ISSIGNED, 5, 5, -1);
    // Give the fixed-point number value 0.96875
    fixed2 = FixedLiteral(FIXED_ISSIGNED, 5, 5, 0.96875);
    // Or these numbers
    fixed3 = FixedOr(fixed1, fixed2);
}
```

The result is -0.03125. This is stored as:

- fixed3.FixedIntBits = -1;
- fixed3.FixedFracBits = 31;

### 1.4.24 FixedRightShift

**FixedRightShift** (Fixed, Shift);

**Arguments**

- **Fixed**: Fixed-point structure of any type and width
- **Shift**: Number of bits to shift right by

**Return values**

Fixed-point number of same type and width as Fixed.

**Description**

Returns Fixed shifted right by Shift number of bits. The number returned is of the same width as Fixed so any bits shifted outside this width are lost.
When shifting unsigned values, the right shift pads the upper bits with zeros. When shifting signed values, the upper bits are copies of the top bit of the original value. Thus a shift right by 1 divides the value by 2 and preserves the sign.

The `Shift` expression must be unsigned and of width

\[
\text{shiftWidth} = \log_2 \text{ceil} (\text{intWidth} + \text{fracWidth} + 1)
\]

where `intWidth` is `width(Fixed.FixedIntBits)` and `fracWidth` is `width(Fixed.FixedFracBits)` (as defined in `FIXED_SIGNED` and `FIXED_UNSIGNED`)

`Shift` has the range: 0 to \(2^{(\text{shiftWidth})} - 1\).

If `Shift` is 0 no shift takes place. If `Shift` is \((\text{intWidth} + \text{fracWidth})\) all the bits are shifted out of `Fixed`.

**Requirements**

Header file: fixed.hch

Library file: fixed.hcl

**Example**

This example shows right shifting on a `FIXED_SIGNED(4, 4)`.

```c
#include <fixed.hch>
set clock = external "P1";
typedef FIXED_SIGNED(4, 4) MyFixed;

void main(void)
{
    MyFixed fixed1, fixed2;
    // Give the fixed-point number value -0.75
    fixed1 = FixedLiteralFromInts(FIXED_ISSIGNED, 4, 4, -1, 4);
    // Shift this number right by 1 bit
    fixed2 = FixedRightShift(fixed1, 1);
}
```

The result is \(-0.375\). This is stored as:

```c
fixed2.FixedIntBits = -1;
fixed2.FixedFracBits = 10;
```
1.4.25 FixedSub

FixedSub( Fixed1, Fixed2 );

Arguments

Fixed1 Fixed-point structure of any type and
1 width

Fixed2 Fixed-point structure of the same type
2 and width

Return values

Fixed-point number of the same type and width as Fixed1 and Fixed2.

Description

Returns Fixed2 subtracted from Fixed1. The number returned is of the same width as
Fixed1 so any bits outside this width are lost.

Requirements

Header file: fixed.hch

Library file: fixed.hcl

Example

This example shows subtraction on a FIXED_SIGNED(2, 4).

#include <fixed.hch>
set clock = external "P1";
typedef FIXED_SIGNED(2, 4) MyFixed;

void main(void)
{
    MyFixed fixed1, fixed2, fixed3;
    // Give the fixed-point number value 1.0625
    fixed1 = FixedLiteral(FIXED_ISSIGNED, 2, 4, 1.0625);
    // Give the fixed-point number value 1.125
    fixed2 = FixedLiteral(FIXED_ISSIGNED, 2, 4, 1.125);
    // Subtract fixed2 from fixed1
    fixed3 = FixedSub(fixed1, fixed2);
}

The result is –0.0625. This is stored as:

fixed2.FixedIntBits = -1;
fixed2.FixedFracBits = 15;
1.4.26 FixedToBits

FixedToBits( Fixed );

Arguments

Fixed - Fixed-point structure of any type and width

d - Integer with type the same as Fixed and width of the sum of the widths of the integer and fraction parts of Fixed.

Description

Returns the integer and fraction parts of Fixed concatenated together.

Requirements

Header file: fixed.hch
Library file: fixed.hcl

Example

This example extracts the bits of a FIXED_UNSIGNED(4, 4).

```
#include <fixed.hch>
set clock = external "P1";
typedef FIXED_UNSIGNED(4, 4) MyFixed;

void main(void)
{
    unsigned int 8 result;
    MyFixed fixed;
    // Assign the value 12.125
    fixed = FixedLiteral(FIXED_ISUNSIGNED, 4, 4, 12.125);
    // Find the type
    result = FixedToBits(fixed);
}
```

The result is equal to $12 \times 2^4 + 2 = 194$:

result = 0b11000010; // binary number
1.4.27 FixedToBool

FixedToBool( Fixed );

Arguments

Fixed-point structure of any type and width

Return values

Single bit wide integer with 0 as false and 1 as true.

Description

Returns 0 if the integer and fraction values of Fixed are equal to zero and 1 otherwise. FixedToBool(x) is equivalent to FixedNEq(x, Zero), where Zero is a fixed-point expression of value 0, and type the same as x.

Requirements

Header file: fixed.hch
Library file: fixed.hcl

Example

This example tests for not equal to 0 of a FIXED_UNSIGNED(4, 4).

#include <fixed.hch>
set clock = external "P1";
typedef FIXED_UNSIGNED(4, 4) MyFixed;

void main(void)
{
    unsigned int 1 result;
    MyFixed fixed;
    // Assign the value 8
    fixed = FixedLiteralFromInts(FIXED_IS_UNSIGNED, 4, 4, 8, 0);
    // Find the type
    result = FixedToBool(fixed);
}

The result is true:

result = 1;
1.4.28 FixedToInt

FixedToInt( Fixed );

Arguments
Fixe Fixed-point structure of any type
d and width

Return values
Integer of same type and width as the integer part of the fixed-point structure.

Description
Returns the integer part of the fixed-point number, rounded towards minus infinity.

Note that this behaviour is different from ISO-C, which rounds towards 0.

Requirements
Header file: fixed.hch
Library file: fixed.hcl

Example
This example extracts the integer part of a FIXED_SIGNED(16, 8).

#include <fixed.hch>
set clock = external "P1";
typedef FIXED_SIGNED(16, 8) MyFixed;

void main(void)
{
    signed int 16 result;
    MyFixed fixed;
    // Assign the value 32767.5
    fixed = FixedLiteral(FIXED_ISSIGNED, 16, 8, 32767.5);
    // Find the integer part of the fixed-point number
    result = FixedToInt(fixed);
}

The result is 32767.
1.4.29 FixedXor

FixedXor( Fixed1, Fixed2 );

Arguments

Fixed1 Fixed-point structure of any type and width
Fixed2 Fixed-point structure of the same type and width

Return values

Fixed-point number of the same type and width as Fixed1 and Fixed2.

Description

Returns bitwise XOR of Fixed1 and Fixed2.

Requirements

Header file: fixed.hch
Library file: fixed.hcl

Example

This example finds the bitwise XOR of two FIXED_SIGNED(5, 5).

```c
#include <fixed.hch>
set clock = external "P1";
typedef FIXED_SIGNED(5, 5) MyFixed;

void main(void)
{
    MyFixed fixed1, fixed2, fixed3;
    // Give the fixed-point number value -1
    fixed1 = FixedLiteral(FIXED_ISSIGNED, 5, 5, -1);
    // Give the fixed-point number value 0.96875
    fixed2 = FixedLiteral(FIXED_ISSIGNED, 5, 5, 0.96875);
    // Xor these numbers
    fixed3 = FixedXor(fixed1, fixed2);
}
```

The result is –1.03125. This is stored as:

```c
fixed3.FixedIntBits = -2;
fixed3.FixedFracBits = 31;
```
2 Index

<
(less than) .................................. 30
<= (less than or equal) .................... 31
>
(greater than) .............................. 23
>= (greater than or equal) .............. 24
A
absolute values ................................ 13
  fixed-point ................................. 13
addition ........................................ 14
  fixed-point .................................. 14
arithmetic operators 13, 14, 19, 20, 32,
  33, 34, 40
  fixed-point 13, 14, 19, 20, 32, 33, 40
B
bit manipulation
  fixed-point operators .................... 26, 38
  bitwise logical operators 15, 36, 37, 44
    bitwise AND ................................ 15
    bitwise NOT ................................ 36
    bitwise OR .................................. 37
    bitwise XOR .............................. 44
boolean type ................................. 42
C
casting ........................................ 16, 17
  fixed-point numbers ...................... 16, 17
conversion ................................. 8, 9, 11
  negative numbers to signed fixed-
    point numbers ................................ 9
  positive numbers to signed fixed-point
    numbers ....................................... 8
  unsigned numbers to fixed-point
    structures .................................. 11
conversion operators 16, 17, 22, 25, 27,
  29, 41, 42, 43
  casting ................................. 16, 17
    width ................................ 22, 25
D
division ....................................... 19, 20
  fixed-point .............................. 19, 20
does not equal ............................. 35
E
equality ....................................... 21, 35
F
  fixed.h ........................................ 4
  fixed.hch ..................................... 4
  fixed.hcl ..................................... 4
  fixed.lib ..................................... 4
  FIXED_ISSIGNED ........................... 5
  FIXED_ISUNSIGNED ......................... 6
  FIXED_SIGNED .............................. 7
  FIXED UNSIGNED ............................ 10
  FixedAbs ................................... 13
  FixedAdd ................................... 14
  FixedAnd ................................... 15
  FixedCastSigned ............................ 16
  FixedCastUnsigned .......................... 17
  FixedDivSigned ............................. 19
  FixedDivUnsigned ........................... 20
  FixedEq ..................................... 21
  FixedFracWidth ............................. 22
  FixedGT .................................... 23
  FixedGTE ................................... 24
  FixedIntWidth .............................. 25
  FixedLeftShift ............................ 26
  FixedLiteral ............................... 27
  FixedLiteralFromInts ..................... 29
  FixedLT .................................... 30
  FixedLTE ................................... 31
  FixedMultSigned ........................... 32
  FixedMultUnsigned ........................ 33