Platform Developer's Kit

Standard Library Manual

The product described in this document is subject to continuous development and improvement. All particulars of the product and its use contained in this document are given by Celoxica Limited in good faith. However, all warranties implied or express, including but not limited to implied warranties of merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. Celoxica Limited shall not be liable for any loss or damage arising from the use of any information in this document, or any incorrect use of the product.

The information contained herein is subject to change without notice and is for general guidance only.

Authors: RG

Document number: RM-1020-1.0

Customer Support at http://www.celoxica.com/support/

Celoxica in Europe	Celoxica in Japan	Celoxica in the Americas
T: +44 (0) 1235 863 656	T: +81 (0) 45 331 0218	T: +1 800 570 7004
E: sales.emea@celoxica.com	E: sales.japan@celoxica.com	E: sales.america@celoxica.com

Contents

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:

```
void main();
```

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

copy ***SourceFileName DestinationFileName***

Optional elements are enclosed in square brackets like this:

```
struct [type_Name]
```

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

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 Standard library (stdlib.hch)

1.1 Introduction: standard macros

The Platform Developer's Kit contains a standard Handel-C library (stdlib.hcl) and header file containing a collection of useful macro expressions. The header file may be used by including it in your Handel-C program with the following line:

```
#include <stdlib.hch>
```

Note that this header file is not the same as the conventional C stdlib.h header file but contains a standard collection of definitions useful to the Handel-C programmer. You must add the module (stdlib.hcl) to the Object\library modules used by the linker to access these macros. This is done in the Linker tab of the Project Settings dialog in the DK GUI.

1.2 Constant definitions

The stdlib.hch header file contains the following constant definitions:

Constant name	Definition
TRUE	1
FALSE	0
NULL	(void*)0

These definitions often lead to cleaner and more readable code. For example:

```
int 8 x with { show=FALSE };

while (TRUE)
{
    ...
}

if (a==TRUE)
{
    ...
}
```

1.3 Bit manipulation macros

The `stdlib.hch` header file contains a number of macro expressions used to manipulate bits and bit fields listed below.

- `adjs`: adjusts width of signed expression
- `adju`: adjusts width of unsigned expression
- `bitrev`: reverses the bits in an expression
- `copy`: duplicates expressions
- `lmo`: finds the position of the most significant 1 bit. (Use `lmo_nz()` if you know that the expression to evaluate is not equal to zero.)
- `lmz`: finds the position of the most significant 0 bit. (Use `lmz_no()` if you know that expression to evaluate is not a number containing all ones. for example, a 4-bit unsigned with value of 15 contains all ones.)
- `population`: counts the number of 1 bits (population) in ***`Expression`***
- `rmo`: finds the position of the least significant 1 bit
- `rmz`: finds the position of the least significant 0 bit
- `top`: extracts the most significant ***`Width`*** bits

1.3.1 adjs macro

Usage: adjs(***Expression***, ***Width***)

Parameters:

Expression Expression to adjust (must be signed integer)

Width Width to adjust to (must be constant)

Returns: Signed integer of width ***Width***.

Description: Adjusts width of signed expression up or down. Sign extends MSBs of expression when expanding width. Drops MSBs of expression when reducing width.

Requirements: Header file: stdlib.hch
Library module: stdlib.hcl

Example:

```
int 4 x;
int 5 y;
int 6 z;

y = 15;
x = adjs(y, width(x)); // x = -1
y = -4;
z = adjs(y, width(z)); // z = -4
```

1.3.2 adju macro

Usage: adju(***Expression***, ***Width***)

Parameters:

Expression	Expression to adjust (must be unsigned integer)
Width	Width to adjust to (must be constant)

Returns: Unsigned integer of width ***Width***.

Description: Adjusts width of unsigned expression up or down. Zero pads MSBs of expression when expanding width. Drops MSBs of expression when reducing width.

Requirements: Header file: stdlib.hch
Library module: stdlib.hcl

Example:

```
unsigned 4 x;
unsigned 5 y;
unsigned 6 z;

y = 14;
x = adju(y, width(x)); // x = 14
z = adju(y, width(z)); // z = 14
```

1.3.3 bitrev macro

Usage: bitrev(***a***)

Parameters:

a	Non-constant expression

Returns: Returns a bit reversed version of parameter ***a***.
For example, 0b0001011 becomes 0b1101000.

Description: Reverses the bits in an expression.

Requirements: Header file: stdlib.hch
Library module: stdlib.hcl

Example:

```
unsigned 8 a, b;
a = 0x34;       // binary = 0b00110100
b = bitrev(a); // b = 0b00101100, 0x26
```

1.3.4 copy macro

Usage: copy(***Expression***, ***Count***)

Parameters:

Expression	Expression to copy
Count	Number of times to copy (constant)

Returns: Expression duplicated ***Count*** times.

Returned expression is of same type as ***Expression***.

Returned width is ***Count*** * width(***Expression***).

Description: Duplicates expressions multiple times.

Requirements: Header file: stdlib.hch
Library module: stdlib.hcl

Example:

```
unsigned 32 x;
unsigned 4 y;

y = 0xA;
x = copy(y, 8); // x = 0xAAAAAAAA
```

1.3.5 lmo macro

Usage: `lmo(` ***Expression*** `)`

Parameters: ***Expression*** Expression to calculate left most one of

Returns: Bit position of leftmost one in ***Expression*** or `width(`***Expression***`)` if ***Expression*** is zero.

The width of the return value is `log2ceil(width(`***Expression***`)+1)` bits.

Description: Finds the position of the most significant 1 bit in an expression.

Requirements: Header file: `stdlib.hch`
Library module: `stdlib.hcl`

Example:

```
int 8 x;
unsigned 4 y;

x = 27;
y = lmo(x); // y = 4
x = 0;
y = lmo(x); // y = 8
```

1.3.6 lmo_nz macro

Usage: `lmo_nz(` ***`Expression`*** `)`

Parameters: ***`Expression`*** Expression to calculate left most one of

Returns: Bit position of leftmost one in ***`Expression`***. If ***`Expression`*** is zero, the result is undefined.

The width of the return value is `log2ceil(width(`***`Expression`***`))` bits.

Description: Finds the position of the most significant 1 bit in a non-zero expression.

`lmo_nz()` produces slightly smaller hardware than `lmo()`, but you can only use if you know that ***`Expression`*** is not equal to zero.

Requirements: Header file: `stdlib.hch`
Library module: `stdlib.hcl`

Example:

```
int 8 x;
unsigned 3 y;

x = 27;
y = lmo_nz(x); // y = 4
x = 0;
y = lmo_nz(x); // value of y is undefined
```

1.3.7 lmz macro

Usage: lmz(***Expression***)

Parameters: ***Expression*** Expression to calculate left most zero of

Returns: Bit position of leftmost zero in ***Expression*** or width(***Expression***) if ***Expression*** is all ones.

The width of the return value is log2ceil(width(***Expression***)+1) bits.

Description: Finds the position of the most significant 0 bit in an expression.

Requirements: Header file: stdlib.hch
Library module: stdlib.hcl

Example:

```
int 8 x;
unsigned 4 y;

x = 27;
y = lmz(x); // y = 7
x = -1;
y = lmz(x); // y = 8
```

1.3.8 lmz_no macro

Usage: `lmz_no(` ***`Expression`*** `)`

Parameters: ***`Expression`*** Expression to calculate left most zero of

Returns: Bit position of leftmost zero in ***`Expression`***. If ***`Expression`*** is all 1's, the result is undefined.

The width of the return value is `log2ceil(width(`***`Expression`***`))` bits.

Description: Finds the position of the most significant 0 in a non "all-ones" expression.

`lmz_no()` produces slightly smaller hardware than `lmz()`, but you can only use it if you know that the value of ***`Expression`*** is not a "all 1's" number. (For example, a 4-bit unsigned variable with a value of 15 is an "all 1's" number since it is stored as 1111.)

Requirements: Header file: `stdlib.hch`
Library module: `stdlib.hcl`

Example:

```
int 8 x;
unsigned 3 y;

x = 27;
y = lmz_no(x); // y = 7
x = -1;
y = lmz_no(x); // value of y is undefined
```

1.3.9 population macro

Usage: `population(` ***`Expression`*** `)`

Parameters: ***`Expression`*** Expression to calculate population of

Returns: Unsigned integer of width `log2ceil(width(`***`Expression`*** `+1))`.

Description: Counts the number of 1 bits (population) in ***`Expression`***.

Requirements: Header file: `stdlib.hch`
Library module: `stdlib.hcl`

Example:

```
int 4 x;
unsigned 3 y;

x = 0b1011;
y = population(x); // y = 3
```

1.3.10 rmo macro

Usage: `rmo(` ***Expression*** `)`

Parameters: ***Expression*** Expression to calculate right most one.

Returns: Bit position of rightmost one in ***Expression*** or `width(`***Expression***`)` if ***Expression*** is zero.

The width of the return value is (`log2ceil(width(`***Expression***`)+1`) bits.

Description: Finds the position of the least significant 1 bit in an expression.

Requirements: Header file: `stdlib.hch`
Library module: `stdlib.hcl`

Example:

```
int 8 x;
unsigned 4 y;

x = 26;
y = rmo(x); // y = 1
x = 0;
y = rmo(x); // y = 8
```

1.3.11 rmz macro

Usage: `rmz(` ***`Expression`*** `)`

Parameters: ***`Expression`*** Expression to calculate right-most zero of.

Returns: Bit position of rightmost zero in ***`Expression`*** or `width(`***`Expression`***`)` if ***`Expression`*** is all ones.

The width of the return value is is `(log2ceil(width(`***`Expression`***`))+1)` bits.

Description: Finds the position of the least significant 0 bit in an expression.

Requirements: Header file: `stdlib.hch`
Library module: `stdlib.hcl`

Example:

```
unsigned 8 x;
unsigned 4 y;

x = 27;
y = rmz(x); // y = 2
x = -1;
y = rmz(x); // y = 8
```

1.3.12 top macro

Usage:	top(***Expression***, ***Width***)	
Parameters:	***Expression***	Expression to extract bits from.
	Width	Number of bits to extract (constant).
Returns:	Value of same width as ***Width***.	
Description:	Extracts the most significant ***Width*** bits from an expression.	
Requirements:	Header file: stdlib.hch Library module: stdlib.hcl	

Example:

```
int 32 x;
int 8 y;

x = 0x12345678;
y = top(x, width(y)); // y = 0x12
```

1.4 Arithmetic macros

The stdlib.hch header file contains a number of macro expressions for mathematical calculations.

abs: absolute value of an expression

- addsat: saturated addition of expressions
- decode: returns ***2***$^{\text{Expression}}$
- div: integer value of ***Expression1***/***Expression2***
- exp2: calculates ***2***$^{\text{Constant}}$
- incwrap: increments value with wrap around at second value
- is_signed: determines the sign of an expression
- log2ceil: calculates $\log_2$ of a number and rounds the result up
- log2floor: calculates $\log_2$ of a number and rounds the result down
- mod: returns remainder of ***Expression1*** divided by ***Expression2***
- sign: gives the sign of an expression (0 positive, 1 negative)
- subsat: saturated subtraction of ***Expression2*** from ***Expression1***
- signed_fast_ge, unsigned_fast_ge: returns one if ***Expression1*** is greater than or equal to ***Expression2***, otherwise zero

- signed_fast_gt , unsigned_fast_gt: returns one if ***Expression1*** is greater than ***Expression2***, otherwise zero
- signed_fast_le , unsigned_fast_le: returns one if ***Expression1*** is less than or equal to ***Expression2***, otherwise zero
- signed_fast_lt , unsigned_fast_lt: returns one if ***Expression1*** is less than ***Expression2***, otherwise zero

1.4.1 abs macro

Usage:	abs(***Expression***)	
Parameters:	***Expression***	Signed expression to get absolute value of
Returns:	Signed value of same width as ***Expression***	
Description:	Obtains the absolute value of an expression	
Requirements:	Header file: stdlib.hch Library module: stdlib.hcl	

Example:

```
int 8 x;
int 8 y;

x = 34;
y = -18;
x = abs(x); // x = 34
y = abs(y); // y = 18
```

1.4.2 addsat macro

Usage: addsat(***Expression1***, ***Expression2***)

Parameters:

Expression1 Unsigned operand 1

Expression2 Unsigned operand 2. Must be of same width as ***Expression1***.

Returns: Unsigned value of same width as ***Expression1*** and ***Expression2***.

Description: Returns sum of ***Expression1*** and ***Expression2***. Addition is saturated and result will not be greater than maximum value representable in the width of the result.

Requirements: Header file: `stdlib.hch`
Library module: `stdlib.hcl`

Example:

```
unsigned 8 x;
unsigned 8 y;
unsigned 8 z;

x = 34;
y = 18;
z = addsat(x, y); // z = 52
x = 34;
y = 240;
z = addsat(x, y); // z = 255
```

1.4.3 decode macro

Usage: decode(***Expression***)

Parameters: ***Expression*** Unsigned operand

Returns: Unsigned value of width $2^{\text{width (Expression)}}$

Description: Returns $\boldsymbol{2}^{\text{Expression}}$

Requirements: Header file: `stdlib.hch`
Library module: `stdlib.hcl`

Example:

```
unsigned 4 x;
unsigned 16 y;

x = 8;
y = decode(x); // y = 0b100000000
```

1.4.4 div macro

Parameters: ***Expression1*** Operand 1

Expression2 Unsigned operand 2. Must be of the same width and type as ***Expression1***.

Returns: Value of same width and type as ***Expression1*** and ***Expression2***.

Description: Returns integer value of ***Expression1***/***Expression2***.

Requirements: Header file: `stdlib.hch`
Library module: `stdlib.hcl`

This macro remains for compatibility with previous versions of Handel-C. Its use is deprecated; use / (division operator) instead.

> Division requires a large amount of hardware and should be avoided unless absolutely necessary.

1.4.5 exp2 macro

Usage: `exp2(` ***Constant*** `)`

Parameters: ***Constant*** Operand.

Returns: Constant of width ***Constant***+1.

Description: Used to calculate $2^{Constant}$. Similar to decode but may be used with constants of undefined width.

Requirements: Header file: `stdlib.hch`
Library module: `stdlib.hcl`

Example:

```
unsigned 4 x;
unsigned (exp2(width(x))) y; // y of width 16
```

1.4.6 incwrap macro

Usage: incwrap(***Expression1***, ***Expression2***)

Parameters:

Expression1	Operand 1
Expression2	Operand 2. Must be of same width as ***Expression1***.

Returns: Value of same type and width as ***Expression1*** and ***Expression2***.

Description: Used to increment a value with wrap around at a second value. Returns 0 if ***Expression1*** equals ***Expression2***, or ***Expression1***+1 otherwise.

Requirements: Header file: stdlib.hch
Library module: stdlib.hcl

Example:

```
unsigned 8 x;

x = 74;
x = incwrap(x, 76); // x = 75
x = incwrap(x, 76); // x = 76
x = incwrap(x, 76); // x = 0
x = incwrap(x, 76); // x = 1
```

1.4.7 is_signed macro

Usage: is_signed(***e***)

Parameters:

e	Non-constant expression

Returns: Returns 1 if ***e*** is signed, 0 if ***e*** is unsigned.

Description: Determines the sign of an expression

Requirements: Header file: stdlib.hch
Library module: stdlib.hcl

Example:

```
unsigned 8 a;
signed 6 b;
unsigned 1 Result;
Result = is_signed(a); // Result == 0
Result = is_signed(b); // Result == 1
```

1.4.8 log2ceil macro

Usage: `log2ceil(` ***`Constant`*** `)`

Parameters: ***`Constant`*** Operand

Returns: Constant value of ceiling(log$_2$(***`Constant`***)).

Description: Used to calculate log$_2$ of a number and rounds the result up. Useful to determine the width of a variable needed to contain a particular value.

Requirements: Header file: `stdlib.hch`
Library module: `stdlib.hcl`

Example:

```
unsigned (log2ceil(5768)) x; // x 13 bits wide
unsigned 8 y;

y = log2ceil(8); // y = 3
y = log2ceil(7); // y = 3
```

1.4.9 log2floor macro

Usage: `log2floor(` ***`Constant`*** `)`

Parameters: ***`Constant`*** Operand

Returns: Constant value of floor(log$_2$(***`Constant`***)).

Description: Used to calculate log$_2$ of a number and rounds the result down.

Requirements: Header file: `stdlib.hch`
Library module: `stdlib.hcl`

Example:

```
unsigned 8 y;

y = log2floor(8); // y = 3
y = log2floor(7); // y = 2
```

1.4.10 mod macro

Parameters: ***Expression1*** Operand 1
Expression2 Operand 2. Must be of the same width and type as ***Expression1***.

Returns: Value of same width and type as ***Expression1*** and ***Expression2***.

Description: Returns remainder of ***Expression1*** divided by ***Expression2***.

Requirements: Header file: `stdlib.hch`
Library module: `stdlib.hcl`

This macro remains for compatibility with previous versions of Handel-C. Its use is deprecated; use % (modulo operator) instead.

> Warning! Modulo arithmetic requires a large amount of hardware and should be avoided unless absolutely necessary.

1.4.11 sign macro

Usage: `sign(` ***Expression*** `)`

Parameters: ***Expression*** Signed operand.

Returns: Unsigned integer 1 bit wide.

Description: Used to obtain the sign of an expression. Returns zero if ***Expression*** is positive or one if ***Expression*** is negative.

Requirements: Header file: `stdlib.hch`
Library module: `stdlib.hcl`

Example:

```
int 8 y;
unsigned 1 z;

y = 53;
z = sign(y); // z = 0
y = -53;
z = sign(y); // z = 1
```

1.4.12 signed_fast_ge macro

Usage:	signed_fast_ge(***Expression1***, ***Expression2***)	
Parameters:	***Expression1***	Signed operand 1
	Expression2	Signed operand 2. Must be of same width as ***Expression1***.
Returns:	Unsigned integer 1 bit wide.	
Description:	Returns one if ***Expression1*** is greater than or equal to ***Expression2***, otherwise zero.	
Requirements:	Header file: stdlib.hch Library module: stdlib.hcl	

Example:

```
int 8 x;
int 8 y;
unsigned 1 z;

x = 100;
y = -100;
z = signed_fast_ge(x, y); // z = 1
x = -15;
y = -15;
z = signed_fast_ge(x, y); // z = 1
```

The signed_fast_ge() macro is deprecated. It may be faster than the Handel-C >= operator for some older FPGA/PLD architectures, such as the Xilinx Spartan, but it is likely to be slower for newer devices.

1.4.13 signed_fast_gt macro

Usage:	`signed_fast_gt(` ***`Expression1`***`,` ***`Expression2`***`)`	
Parameters:	***Expression1***	Signed operand 1
	Expression2	Signed operand 2. Must be of same width as ***Expression1***.
Returns:	Unsigned integer 1 bit wide.	
Description:	Returns one if ***Expression1*** is greater than ***Expression2***, otherwise zero.	
Requirements:	Header file: `stdlib.hch` Library module: `stdlib.hcl`	

Example:

```
int 8 x;
int 8 y;
unsigned 1 z;

x = 100;
y = -100;
z = signed_fast_gt(x, y); // z = 1
x = -15;
y = -15;
z = signed_fast_gt(x, y); // z = 0
```

The `signed_fast_gt()` macro is deprecated. It may be faster than the Handel-C > operator for some older FPGA/PLD architectures, such as the Xilinx Spartan, but it is likely to be slower for newer devices.

1.4.14 signed_fast_le macro

Usage: signed_fast_le(***Expression1***, ***Expression2***)

Parameters:

Expression1	Signed operand 1
Expression2	Signed operand 2. Must be of same width as ***Expression1***.

Returns: Unsigned integer 1 bit wide.

Description: Returns one if ***Expression1*** is less than or equal to ***Expression2***, otherwise zero.

Requirements: Header file: stdlib.hch
Library module: stdlib.hcl

Example:

```
int 8 x;
int 8 y;
unsigned 1 z;

x = -20;
y = 111;
z = signed_fast_le(x, y); // z = 1
x = -15;
y = -15;
z = signed_fast_le(x, y); // z = 1
```

The signed_fast_le() macro is deprecated. It may be faster than the Handel-C <= operator for some older FPGA/PLD architectures, such as the Xilinx Spartan, but it is likely to be slower for newer devices.

1.4.15 signed_fast_lt macro

Usage: `signed_fast_lt(` ***`Expression1`***, ***`Expression2`*** `)`

Parameters:

`Expression1` Signed operand 1

`Expression2` Signed operand 2. Must be of same width as ***`Expression1.`***

Returns: Unsigned integer 1 bit wide.

Description: Returns one if ***`Expression1`*** is less than ***`Expression2`***, otherwise zero.

Requirements: Header file: `stdlib.hch`
Library module: `stdlib.hcl`

Example:

```
int 8 x;

int 8 y;
unsigned 1 z;

x = -57;
y = -22;
z = signed_fast_lt(x, y); // z = 1
x = -15;
y = -15;
z = signed_fast_lt(x, y); // z = 0
```

> The `signed_fast_lt()` macro is deprecated. It may be faster than the Handel-C < operator for some older FPGA/PLD architectures, such as the Xilinx Spartan, but it is likely to be slower for newer devices.

1.4.16 subsat macro

Usage: `subsat(` ***`Expression1`***, ***`Expression2`*** `)`

Parameters:

`Expression1` Unsigned operand 1.

`Expression2` Unsigned operand 2. Must be of same width as ***`Expression1`***.

Returns: Unsigned value of same width as ***`Expression1`*** and ***`Expression2`***.

Description: Returns result of subtracting ***`Expression2`*** from ***`Expression1`***. Subtraction is saturated and result will not be less than 0.

Requirements: Header file: `stdlib.hch`
Library module: `stdlib.hcl`

Example:

```
unsigned 8 x;
unsigned 8 y;
unsigned 8 z;

x = 34;
y = 18;
z = subsat(x, y); // z = 16
x = 34;
y = 240;
z = subsat(x, y); // z = 0
```

1.4.17 unsigned_fast_ge macro

Usage: `unsigned_fast_ge(` ***Expression1***, ***Expression2*** `)`

Parameters:

Expression1	Unsigned operand 1
Expression2	Unsigned operand 2. Must be of same width as ***Expression1***.

Returns: Unsigned integer 1 bit wide.

Description: Returns one if ***Expression1*** is greater than or equal to ***Expression2***, otherwise zero.

Requirements: Header file: `stdlib.hch`
Library module: `stdlib.hcl`

Example:

```
unsigned 8 x;
unsigned 8 y;
unsigned 1 z;

x = 231;
y = 198;
z = unsigned_fast_ge(x, y); // z = 1
x = 155;
y = 155;
z = unsigned_fast_ge(x, y); // z = 1
```

✶ The unsigned_fast_ge() macro is deprecated. It may be faster than the Handel-C >= operator for some older FPGA/PLD architectures, such as the Xilinx Spartan, but it is likely to be slower for newer devices.

1.4.18 unsigned_fast_gt macro

Usage: `unsigned_fast_gt(` ***Expression1***, ***Expression2*** `)`

Parameters:

Expression1 Unsigned operand 1

Expression2 Unsigned operand 2. Must be of same width as ***Expression1***.

Returns: Unsigned integer 1 bit wide.

Description: Returns one if ***Expression1*** is greater than ***Expression2***, otherwise zero.

Requirements: Header file: `stdlib.hch`
Library module: `stdlib.hcl`

Example:

```
unsigned 8 x;
unsigned 8 y;
unsigned 1 z;

x = 231;
y = 198;
z = unsigned_fast_gt(x, y); // z = 1
x = 155;
y = 155;
z = unsigned_fast_gt(x, y); // z = 0
```

The `unsigned_fast_gt()` macro is deprecated. It may be faster than the Handel-C > operator for some older FPGA/PLD architectures, such as the Xilinx Spartan, but it is likely to be slower for newer devices.

1.4.19 unsigned_fast_le macro

Usage:	`unsigned_fast_le(` ***`Expression1`***`,` ***`Expression2`***`)`	
Parameters:	***Expression1***	Unsigned operand 1
	Expression2	Unsigned operand 2. Must be of same width as ***Expression1***.
Returns:	Unsigned integer 1 bit wide.	
Description:	Returns one if ***Expression1*** is less than or equal to ***Expression2***, otherwise zero.	
Requirements:	Header file: `stdlib.hch` Library module: `stdlib.hcl`	

Example:

```
unsigned 8 x;
unsigned 8 y;
unsigned 1 z;

x = 162;
y = 198;
z = unsigned_fast_le(x, y);  // z = 1
x = 155;
y = 155;
z = unsigned_fast_le(x, y);  // z = 1
```

✷ The `unsigned_fast_le()` macro is deprecated. It may be faster than the Handel-C <= operator for some older FPGA/PLD architectures, such as the Xilinx Spartan, but it is likely to be slower for newer devices.

1.4.20 unsigned_fast_lt macro

Usage: `unsigned_fast_lt(` ***`Expression1`***, ***`Expression2`*** `)`

Parameters:

`Expression1`	Unsigned operand 1
`Expression2`	Unsigned operand 2. Must be of same width as ***`Expression1`***.

Returns: Unsigned integer 1 bit wide.

Description: Returns one if ***`Expression1`*** is less than ***`Expression2`***, otherwise zero.

Requirements: Header file: `stdlib.hch`
Library module: `stdlib.hcl`

Example:

```
unsigned 8 x;
unsigned 8 y;
unsigned 1 z;

x = 162;
y = 198;
z = unsigned_fast_lt(x, y); // z = 1
x = 155;
y = 155;
z = unsigned_fast_lt(x, y); // z = 0
```

> The `unsigned_fast_lt()` macro is deprecated. It may be faster than the Handel-C < operator for some older FPGA/PLD architectures, such as the Xilinx Spartan, but it is likely to be slower for newer devices.

2 Index