Platform Developer’s Kit

Standard Library Manual
Celoxica, the Celoxica logo and Handel-C are trademarks of Celoxica Limited.

All other products or services mentioned herein may be trademarks of their respective owners.

Neither the whole nor any part of the information contained in, or the product described in, this document may be adapted or reproduced in any material form except with the prior written permission of the copyright holder.

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.

Copyright © 2005 Celoxica Limited. All rights reserved.

Authors: RG

Document number: RM-1020-1.0

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

Celoxica in Europe

T: +44 (0) 1235 863 656
E: sales.emea@celoxica.com

Celoxica in Japan

T: +81 (0) 45 331 0218
E: sales.japan@celoxica.com

Celoxica in the Americas

T: +1 800 570 7004
E: sales.america@celoxica.com

www.celoxica.com
Contents

1 STANDARD LIBRARY (STDLIB.HCH) ...................................................... 4
  1.1 INTRODUCTION: STANDARD MACROS........................................... 4
  1.2 CONSTANT DEFINITIONS ................................................................. 4
  1.3 BIT MANIPULATION MACROS .......................................................... 5
    1.3.1 adjs macro .................................................................................. 6
    1.3.2 adju macro ................................................................................ 6
    1.3.3 bitrev macro .............................................................................. 7
    1.3.4 copy macro ................................................................................. 8
    1.3.5 lmo macro .................................................................................. 9
    1.3.6 lmo_nz macro ............................................................................. 9
    1.3.7 lmz macro .................................................................................. 11
    1.3.8 lmz_no macro .......................................................................... 11
    1.3.9 population macro ....................................................................... 12
    1.3.10 rmo macro ............................................................................... 13
    1.3.11 rmz macro ............................................................................... 14
    1.3.12 top macro ............................................................................... 15
  1.4 ARITHMETIC MACROS ................................................................ 15
    1.4.1 abs macro ................................................................................ 16
    1.4.2 addsat macro .......................................................................... 17
    1.4.3 decode macro ........................................................................... 17
    1.4.4 div macro ................................................................................ 18
    1.4.5 exp2 macro .............................................................................. 18
    1.4.6 incwrap macro ......................................................................... 19
    1.4.7 is_signed macro ....................................................................... 19
    1.4.8 log2ceil macro ....................................................................... 20
    1.4.9 log2floor macro ....................................................................... 20
    1.4.10 mod macro .............................................................................. 21
    1.4.11 sign macro ............................................................................. 21
    1.4.12 signed_fast_ge macro ............................................................ 22
    1.4.13 signed_fast_gt macro ............................................................ 23
    1.4.14 signed_fast_le macro ............................................................ 24
    1.4.15 signed_fast_lt macro ............................................................ 25
    1.4.16 subsat macro ......................................................................... 26
    1.4.17 unsigned_fast_ge macro ......................................................... 27
    1.4.18 unsigned_fast_gt macro ......................................................... 28
    1.4.19 unsigned_fast_le macro ......................................................... 29
    1.4.20 unsigned_fast_lt macro ......................................................... 30

2 INDEX ........................................................................................................ 31
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 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:

<table>
<thead>
<tr>
<th>Constant name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>1</td>
</tr>
<tr>
<td>FALSE</td>
<td>0</td>
</tr>
<tr>
<td>NULL</td>
<td>(void*)0</td>
</tr>
</tbody>
</table>

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: \texttt{adjs( Expression, Width )}

Parameters:
- \texttt{Expression}: Expression to adjust (must be signed integer)
- \texttt{Width}: Width to adjust to (must be constant)

Returns: Signed integer of width \texttt{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: \texttt{stdlib.hch}
- Library module: \texttt{stdlib.hcl}

Example:

```c
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: \texttt{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 \texttt{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:

```c
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: \texttt{bitrev} (a)

Parameters: 
- **a**: Non-constant expression

Returns: Returns a bit reversed version of parameter \texttt{a}. For example, 0b0001011 becomes 0b1101000.

Description: Reverses the bits in an expression.

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

Example:

```c
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:
```c
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: \texttt{lmo\_nz}(\textit{Expression})

Parameters: \textit{Expression} Expression to calculate left most one of

Returns: Bit position of leftmost one in \textit{Expression}. If \textit{Expression} is zero, the result is undefined.

The width of the return value is \(\log_2\lceil\text{width}(\textit{Expression})\rceil\) bits.

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

\texttt{lmo\_nz()} produces slightly smaller hardware than \texttt{lmo()}, but you can only use if you know that \textit{Expression} is not equal to zero.

Requirements: Header file: stdlib.hch

Library module: stdlib.hcl

Example:

```c
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 \textit{lmz} macro

\textbf{Usage:}\ 
\texttt{lmz( Expression )}

\textbf{Parameters:}\ 
\textit{Expression} \hspace{1em} Expression to calculate left most zero of

\textbf{Returns:}\ 
Bit position of leftmost zero in \textit{Expression} or \text{width(\textit{Expression})} if \textit{Expression} is all ones.

The width of the return value is \text{log2ceil(width(\textit{Expression})+1)} bits.

\textbf{Description:}\ 
Finds the position of the most significant 0 bit in an expression.

\textbf{Requirements:}\ 
Header file: stdlib.hch
Library module: stdlib.hcl

\textbf{Example:}\ 
\begin{verbatim}
int 8 x;
unsigned 4 y;

x = 27;
y = lmz(x); \// y = 7
x = -1;
y = lmz(x); \// y = 8
\end{verbatim}

1.3.8 \textit{lmz_no} macro
Usage:          \texttt{lmz\_no( Expression )} \\
Parameters:    \texttt{Expression} \quad \text{Expression to calculate left most zero of} \\
Returns:       Bit position of leftmost zero in\texttt{Expression}. If \texttt{Expression} is all 1's, the result is undefined. \\
                The width of the return value is \\
                \texttt{log2ceil(width(\texttt{Expression}))} bits. \\
Description:   Finds the position of the most significant 0 in a non "all-ones" expression. \\
                \texttt{lmz\_no()} produces slightly smaller hardware than \texttt{lmz()}, but you can only use it if you know that the value of \texttt{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:       \\
                \begin{verbatim} 
                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 
                \end{verbatim} 

\subsection*{1.3.9 \textit{population} macro}

Usage:          \texttt{population( Expression )} \\
Parameters:    \texttt{Expression} \quad \text{Expression to calculate population of} \\
Returns:       Unsigned integer of width \texttt{log2ceil(width(\texttt{Expression} +1))}. \\
Description:   Counts the number of 1 bits (population) in \texttt{Expression}. \\
Requirements:  Header file: stdlib.hch \\
                Library module: stdlib.hcl \\
Example:       \\
                \begin{verbatim} 
                int 4 x; 
                unsigned 3 y; 

                x = 0b1011; 
                y = population(x); // y = 3 
                \end{verbatim}
1.3.10 rmo macro

Usage: \[ \text{rmo}\left( \text{Expression} \right) \]

Parameters: \text{Expression} \quad \text{Expression to calculate right most one.}

Returns: Bit position of rightmost one in \text{Expression} or \text{width(\text{Expression})} if \text{Expression} is zero.

The width of the return value is \( \log_2(\text{width(\text{Expression})}+1) \) bits.

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

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

Example:

```c
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:**

```c
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 

\[
\text{(log2ceil(width(Expression))}+1) \text{ bits.}
\]

**Description:**

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

**Requirements:**

- Header file: `stdlib.h`
- Library module: `stdlib.hcl`

**Example:**

```c
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: 

\[
\text{top(} \text{Expression, Width} \text{)}
\]

Parameters: 

- \text{Expression}: Expression to extract bits from.
- \text{Width}: Number of bits to extract (constant).

Returns: 

Value of same width as \text{Width}.

Description: 

Extracts the most significant \text{Width} bits from an expression.

Requirements: 

Header file: stdlib.h
Library module: stdlib.hcl

Example: 

```c
int 32 x;
int 8 y;

x = 0x12345678;
y = \text{top(x, width(y))}; // y = 0x12
```

1.4 Arithmetic macros

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

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

\[ \text{addsat}( \text{Expression1}, \text{Expression2} ) \]

Parameters: 

- \text{Expression1}  
  Unsigned operand 1
- \text{Expression2}  
  Unsigned operand 2. Must be of same width as \text{Expression1}.

Returns: 

Unsigned value of same width as \text{Expression1} and \text{Expression2}.

Description: 

Returns sum of \text{Expression1} and \text{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: 

\[ \text{decode}( \text{Expression} ) \]

Parameters: 

- \text{Expression}  
  Unsigned operand

Returns: 

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

Description: 

Returns \(2^{\text{Expression}}\)

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: \texttt{incwrap( Expression1, Expression2 )}

Parameters: 
- \texttt{Expression1}  Operand 1
- \texttt{Expression2}  Operand 2. Must be of same width as \texttt{Expression1}.

Returns: Value of same type and width as \texttt{Expression1} and \texttt{Expression2}.

Description: Used to increment a value with wrap around at a second value. Returns 0 if \texttt{Expression1} equals \texttt{Expression2}, or \texttt{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: \texttt{is_signed( e )}

Parameters: 
- \texttt{e}  Non-constant expression

Returns: Returns 1 if \texttt{e} is signed, 0 if \texttt{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: 
\[ \text{log2ceil( Constant )} \]

Parameters:  
\text{Constant} \quad \text{Operand}

Returns:  
Constant value of ceiling(\log_2(\text{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 (\text{log2ceil}(5768)) x; // x 13 bits wide
unsigned 8 y;

\[ y = \text{log2ceil}(8); \quad // \ y = 3 \]
\[ y = \text{log2ceil}(7); \quad // \ y = 3 \]

1.4.9 log2floor macro

Usage:  
\[ \text{log2floor( Constant )} \]

Parameters:  
\text{Constant} \quad \text{Operand}

Returns:  
Constant value of floor(\log_2(\text{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;

\[ y = \text{log2floor}(8); \quad // \ y = 3 \]
\[ y = \text{log2floor}(7); \quad // \ y = 2 \]
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:

```c
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:

```c
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:**
```c
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:  

```c
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:

```c
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: 

\[ \text{unsigned\_fast\_ge( Expression1, Expression2 )} \]

Parameters: 
- \( \text{Expression1} \)  
  Unsigned operand 1
- \( \text{Expression2} \)  
  Unsigned operand 2. Must be of same width as \( \text{Expression1} \).

Returns: 
Unsigned integer 1 bit wide.

Description: 
Returns one if \( \text{Expression1} \) is greater than or equal to \( \text{Expression2} \), 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 \text{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: \[ \text{unsigned\_fast\_le}( \text{Expression1}, \text{Expression2} ) \]

Parameters:
- \text{Expression1}  
  Unsigned operand 1
- \text{Expression2}  
  Unsigned operand 2. Must be of same width as \text{Expression1}.

Returns: Unsigned integer 1 bit wide.

Description: Returns one if \text{Expression1} is less than or equal to \text{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 \text{unsigned\_fast\_le()} macro is deprecated. It may be faster than the Handel-C \text{<=} 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:**

```c
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

<table>
<thead>
<tr>
<th>Letter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>abs macro........................................... 16</td>
</tr>
<tr>
<td>A</td>
<td>absolute values..................................... 16</td>
</tr>
<tr>
<td>A</td>
<td>addition.............................................. 17</td>
</tr>
<tr>
<td>A</td>
<td>addsat macro......................................... 17</td>
</tr>
<tr>
<td>A</td>
<td>adjs macro.......................................... 6</td>
</tr>
<tr>
<td>A</td>
<td>adju macro.......................................... 6</td>
</tr>
<tr>
<td>A</td>
<td>adjusting widths.................................... 6</td>
</tr>
<tr>
<td>A</td>
<td>arithmetic macros................................. 4, 15</td>
</tr>
<tr>
<td>B</td>
<td>bit manipulation................................... 4, 5</td>
</tr>
<tr>
<td>C</td>
<td>constant definitions............................... 4</td>
</tr>
<tr>
<td>C</td>
<td>copy macro.......................................... 8</td>
</tr>
<tr>
<td>D</td>
<td>decode macro....................................... 17</td>
</tr>
<tr>
<td>D</td>
<td>div macro.......................................... 18</td>
</tr>
<tr>
<td>D</td>
<td>division............................................. 18</td>
</tr>
<tr>
<td>D</td>
<td>duplication......................................... 8</td>
</tr>
<tr>
<td>E</td>
<td>exp2 macro......................................... 18</td>
</tr>
<tr>
<td>F</td>
<td>FALSE............................................... 4</td>
</tr>
<tr>
<td>H</td>
<td>header files....................................... 4</td>
</tr>
<tr>
<td>H</td>
<td>standards.......................................... 4</td>
</tr>
<tr>
<td>I</td>
<td>increment with wrap................................ 19</td>
</tr>
<tr>
<td>I</td>
<td>incwrap macro...................................... 19</td>
</tr>
<tr>
<td>L</td>
<td>left most one...................................... 9</td>
</tr>
<tr>
<td>L</td>
<td>left most zero..................................... 11</td>
</tr>
<tr>
<td>M</td>
<td>macro expressions................................. 4</td>
</tr>
<tr>
<td>M</td>
<td>macros................................................ 4</td>
</tr>
<tr>
<td>M</td>
<td>mod macro.......................................... 21</td>
</tr>
<tr>
<td>M</td>
<td>modulo arithmetic................................. 21</td>
</tr>
<tr>
<td>N</td>
<td>NULL................................................. 4</td>
</tr>
<tr>
<td>P</td>
<td>population macro................................. 12</td>
</tr>
<tr>
<td>R</td>
<td>right most one.................................... 13</td>
</tr>
<tr>
<td>R</td>
<td>right most zero.................................. 14</td>
</tr>
<tr>
<td>R</td>
<td>rmo macro.......................................... 13</td>
</tr>
<tr>
<td>R</td>
<td>rmz macro.......................................... 14</td>
</tr>
<tr>
<td>S</td>
<td>saturated addition............................... 17</td>
</tr>
<tr>
<td>S</td>
<td>saturated subtraction............................ 26</td>
</tr>
<tr>
<td>S</td>
<td>sign................................................. 21</td>
</tr>
<tr>
<td>S</td>
<td>sign macro........................................... 21</td>
</tr>
<tr>
<td>S</td>
<td>sign extending..................................... 6</td>
</tr>
<tr>
<td>S</td>
<td>signed_fast_ge................................... 22</td>
</tr>
<tr>
<td>S</td>
<td>signed_fast_gt.................................... 23</td>
</tr>
<tr>
<td>S</td>
<td>signed_fast_le.................................... 24</td>
</tr>
<tr>
<td>S</td>
<td>signed_fast_lt.................................... 25</td>
</tr>
<tr>
<td>S</td>
<td>stdlib.hch......................................... 4, 5, 15</td>
</tr>
<tr>
<td>T</td>
<td>top macro......................................... 15</td>
</tr>
<tr>
<td>TRUE</td>
<td>..................................................... 4</td>
</tr>
</tbody>
</table>

[www.celoxica.com](http://www.celoxica.com)
U

unsigned_fast_ge............................ 27
unsigned_fast_gt ............................ 28
unsigned_fast_le............................. 29
unsigned_fast_lt ............................. 30

Z

zero padding ....................................6